การตรวจสอบสภาพพร้อมใช้งานของระบบแบบซอฟท์เรียลไทม์โดยวิธีการวิเคราะห์ล็อกแบบกระจาย

นายปัญญา กิตติพิพัฒนถาวร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

SOFT REAL-TIME AVAILABILITY MONITORING SYSTEM USING DISTRIBUTED LOG ANALYSIS

Mr. Panya Kittipipattanathaworn

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	การตรวจสอบสภาพพร้อมใช้งานของระบบแบบซอฟท์เรียลไทม์โดยวิธีการวิเคราะห์ล็อกแบบกระจาย
โดย	นายปัญญา กิตติพิพัฒนถาวร
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร.บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาศ)

ปัญญา กิตติพิพัฒนถาวร : การตรวจสอบสภาพพร้อมใช้งานของระบบแบบซอฟท์เรียลไทม์โดยวิธีการวิเคราะห์ล็อกแบบกระจาย. (SOFT REAL-TIME AVAILABILITY MONITORING SYSTEM USING DISTRIBUTED LOG ANALYSIS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.ณัฐวุฒิ หนูไพโรจน์, 67 หน้า.

การตรวจสอบข้อมูลแบบเรียลไทม์สามารถนำมาใช้สำหรับตรวจสอบประสิทธิภาพ, สภาพพร้อมใช้งานหรือสิ่งผิดปกติภายในระบบโดยอาจจะสามารถนำมาวิเคราะห์และหาความสัมพันธ์ของข้อมูลจากล็อกไฟล์ แต่อย่างไรก็ตาม ด้วยความซับซ้อนของระบบที่เพิ่มมากขึ้น จำนวนและขนาดของล็อกไฟล์ได้เพิ่มมากขึ้นอย่างมหาศาล ทำให้การตรวจสอบข้อผิดพลาดตามข้อจำกัดทางเวลา (Timing Constraint) กลายเป็นสิ่งสำคัญที่จะกำหนดเงื่อนไขของการการันตีคุณภาพการให้บริการโดยแต่ละระบบอาจมีความต้องการความเป็น Real-time ในระดับที่ต่างกัน ซึ่งสามารถแบ่งได้เป็น Soft และ Hard ขึ้นอยู่กับว่าระบบนั้นๆจะมีผลกระทบต่อการละเมิด deadline ได้มากน้อยเพียงใดโดยสามารถกำหนดให้ออกมาในรูปแบบของข้อตกลงในการให้บริการ (Service Level Agreement – SLAs)

ในงานวิจัยนี้ได้ทำการนิยามโครงสร้างของระบบตรวจสอบแบบเรียลไทม์ (Real-time monitoring) สำหรับระบบซอฟท์เรียลไทม์ (Soft real-time system) โดยอาศัยหลักทางสถิติมาประยุกต์ใช้เป็นแนวทางในการคำนวณโอกาสที่จะละเมิดระยะเวลาที่กำหนด (Deadline) ในระบบ และสามารถคำนวณหาความน่าจะเป็นในการกำหนดระยะเวลาที่เหมาะสม นอกจากนี้ยังพัฒนาโครงสร้างและแก้ไขปัญหาที่เกี่ยวข้องกับระบบตรวจสอบ เพื่อรองรับการทำงานแบบเรียลไทม์ โดยนำสถาปัตยกรรมเชิง Big data มาใช้ในการวิเคราะห์และหาความสัมพันธ์ของ Event log ที่เรียกว่า log correlation เพื่อวิเคราะห์ต้นตอของสาเหตุและระบุปัญหาที่เกิดขึ้นจากข้อมูล log เพื่อแก้ปัญหาที่เกิดขึ้นภายในระบบได้เร็วยิ่งขึ้น

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5470272621 : MAJOR COMPUTER ENGINEERING
KEYWORDS: MONITORING SYSTEM; REAL-TIME; SLAS; SOFT DEADLINE; LOG CORRELATION; AVAILABILITY

PANYA KITTIPIPATTANATHAWORN: SOFT REAL-TIME AVAILABILITY MONITORING SYSTEM USING DISTRIBUTED LOG ANALYSIS. ADVISOR: ASST. PROF. NATAWUT NUPAIROJ, Ph.D., 67 pp.

Real-time monitoring of data streams can be used for monitoring performance, availability or system anomaly by analyzing and correlating events from logs file. However, due to the increasing complexity of modern systems, the size of logs can become very large. Monitoring with timing constraint becomes more important in facilitating the enforcement of conditional guarantees. In general, real-time systems are usually classified into soft and hard which need to comply to the term of Service Level Agreements (SLAs).

In this research, we propose a soft real-time monitoring framework by adopting the statistical approach in order to approximate and calculate the probability of the violation of the deadline constraint. Moreover, we also address the issues in real-time monitoring system. By using Big data architecture for data analysis, we can perform log correlation to analyze root cause and solve the problem more quickly.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Engineering Advisor's Signature .....
Academic Year: 2013

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความกรุณาอย่างสูงจากผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่กรุณาสละเวลาช่วยเหลือในการให้ความรู้ คำปรึกษา คำแนะนำ รวมทั้งให้แนวคิดที่เป็นประโยชน์ในการทำวิจัย และช่วยชี้แนะการแก้ไขปัญหาต่างๆ ที่เกิดขึ้นระหว่างการทำวิจัย ขอขอบพระคุณอาจารย์มา ณ โอกาสนี้ และขอขอบพระคุณคณาจารย์ทุกท่านที่แนะนำสั่งสอนและให้ความรู้แก่ข้าพเจ้าตลอดระยะเวลาการศึกษา

ขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทุกท่านเป็นอย่างสูง ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา ประธานคณะกรรมการสอบวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร. วีระ เหมืองสิน กรรมการสอบวิทยานิพนธ์ และ ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาศ กรรมการสอบวิทยานิพนธ์ ที่ให้ความกรุณาตรวจสอบและให้ข้อเสนอแนะอันเป็นประโยชน์ในการปรับปรุงแก้ไขวิทยานิพนธ์นี้ให้มีความถูกต้องและสมบูรณ์มากยิ่งขึ้น รวมทั้งคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่ได้ให้ความรู้ และให้การอบรมสั่งสอน

ขอกราบขอบพระคุณ นายณรงค์ กิตติพิพัฒนถาวร , นางสาวสุพัฒนา อนุตตรพงศา รวมถึงขอขอบคุณนายปริญญา กิตติพิพัฒนถาวร บิดามารดาและพี่ชายของผู้วิจัยที่ให้การสนับสนุน เอาใจใส่ คอยห่วงใย และให้กำลังใจแก่ผู้วิจัยเสมอมา

นอกจากนี้ ผู้วิจัยขอขอบคุณทุกท่านที่มีส่วนช่วยเหลือจนทำให้วิทยานิพนธ์ฉบับนี้สำเร็จเรียบร้อยลงได้ด้วยดีทุกประการ

สารบัญ

หน้า

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหา

ปัจจุบันองค์กรต่างๆจะมีระบบ IT เป็นพื้นฐานในการทำงานและปัญหาหลักที่ทุกองค์กรต้องรับมือ คือ ความต้องการที่เพิ่มมากขึ้นทั้งในเรื่องของ การประมวลผล และขนาดของข้อมูลที่เพิ่มมากขึ้นเรื่อยๆ (Big Data Crisis) ซึ่งคำว่า Big Data [1] หรือฐานข้อมูลขนาดใหญ่ ในที่นี้หมายถึง เซตของข้อมูลที่ใหญ่มากขึ้นเรื่อยๆและเกิดขึ้นอย่างรวดเร็วอยู่ตลอดเวลา จนยากที่จะใช้เครื่องมือการบริหารที่ทำงานด้วยมือคนแบบที่เคยเป็นมา อีกทั้งไม่สามารถวิเคราะห์ log ได้อย่างทันท่วงที (Real-time) รวมถึงสภาพแวดล้อมในระบบมีความแตกต่างกันและเปลี่ยนแปลงได้ตลอดเวลา จนทำให้เกิดความซับซ้อนมากขึ้นเรื่อยๆภายในระบบ [2] ขนาดและ Format ที่ต่างกันของข้อมูล Log ก็ยากต่อการวิเคราะห์และไม่สามารถตรวจสอบได้ [3] การเสียของอุปกรณ์ภายในระบบเพียงจุดเดียว (Hardware Failure) อาจจะส่งผลให้เกิด Log กระจายไปทุกที่และ event ตามมามหาศาล ทั้งที่ต้นตอของปัญหาอาจเกิดจากจุดเพียงจุดเดียว ซึ่งอาจส่งผลกระทบต่อการให้บริการของของระบบทำให้ไม่สามารถใช้งานได้ (Service Failure) [2, 4], ใช้งานได้ไม่เต็มประสิทธิภาพ (Service Degradation) [5, 6] หรือไม่ส่งผลกระทบต่อการให้บริการเลยก็เป็นได้ ดังนั้นการตรวจสอบและวิเคราะห์ข้อมูล (Monitoring and analyzing) ระบบขนาดใหญ่ในลักษณะที่เป็น Real-time stream processing นั้นจึงเป็นเรื่องสำคัญ

การตรวจสอบข้อมูลแบบเรียลไทม์สามารถนำมาใช้สำหรับตรวจสอบประสิทธิภาพ, สภาพพร้อมใช้งานหรือสิ่งผิดปกติภายในระบบ โดยอาจจะสามารถนำมาวิเคราะห์และหาความสัมพันธ์ของข้อมูลจากล็อกไฟล์ แต่อย่างไรก็ตาม ด้วยความซับซ้อนของระบบที่เพิ่มมากขึ้นในลักษณะที่เป็น Distributed system รวมถึงจำนวนและขนาดของล็อกไฟล์ได้เพิ่มมากขึ้นอย่างมหาศาล ทำให้การตรวจสอบข้อผิดพลาดและสิ่งผิดปกติที่เกิดขึ้นในระบบตามข้อจำกัดทางเวลา (Timing Constraint) กลายเป็นสิ่งสำคัญที่จะกำหนดเงื่อนไขของการการันตีคุณภาพการให้บริการโดยแต่ละระบบอาจมีความต้องการความเป็น Real-time ในระดับที่ต่างกัน [7] ซึ่งโดยปกติแล้วระบบที่เป็น Real-time นั้นสามารถแบ่งออกเป็น 2 ลักษณะ คือ Hard Real-time และ Soft real-time ขึ้นอยู่กับระบบนั้นๆ จะมีผลกระทบต่อการละเมิด deadline ได้มากน้อยเพียงใดโดยสามารถกำหนดให้ออกมาในรูปแบบของข้อตกลงในการให้บริการ (Service Level Agreement – SLAs) โดยระบบที่มีลักษณะเป็น Hard Real-time จะถูกออกแบบให้ไม่สามารถเกิดการละเมิด Deadline ใดๆได้ เนื่องจากหากมีการละเมิด Deadline constraint อาจจะทำให้ระบบไม่สามารถใช้งานได้ (System failure) ยกตัวอย่าง เช่น ระบบ flight control system ในขณะที่ Soft real-time นั้น หากเกิดการละเมิด deadline constraint อาจทำให้ระบบมีประสิทธิภาพลดลง แต่ไม่ถึงกับทำให้ระบบไม่สามารถใช้งานได้ ยกตัวอย่างระบบที่มีลักษณะเป็น Soft real-time เช่น ระบบการตรวจสอบและหาความสัมพันธ์ข้อมูลการใช้บริการ เพื่อเป็นแนวทางในการบริหารจัดการและสร้างความพึงพอใจให้แก่ผู้ใช้บริการ หรือจะเป็นการตรวจสอบระบบในส่วนของการคาดเดาเหตุการณ์ที่จะเกิดขึ้น (Trend prediction)

รวมถึงการตรวจจับสิ่งผิดปกติที่เกิดขึ้นในระบบ เช่น Error ที่เกิดขึ้นในระบบ เป็นต้น ดังนั้นระบบที่มีลักษณะเป็น Soft real-time จะสามารถทนต่อการละเมิด deadline ได้ โดยระบบจะยังคงสามารถใช้งานได้ตราบที่จำนวนของการละเมิด deadline ยังคงอยู่ในกรอบหรือขอบเขตที่ระบบสามารถรับได้ โดยการทำงานในลักษณะนี้จะมีการกำหนดคุณภาพของการให้บริการ (QoS -- Quality of Service) หรือระดับข้อตกลงของการให้บริการ (SLA -- Service level agreement) เช่น 90% ของระบบ Monitoring จะต้องสามารถตรวจจับความผิดพลาด (Detect Error) จาก Event log ให้ได้ภายใน 10 วินาที เป็นต้น

การวิเคราะห์ Service Failure ที่เกิดขึ้นในการให้บริการจึงเป็นเรื่องสำคัญ โดยการวิเคราะห์และวัดผลโดยตรงจากสิ่งที่เกิดขึ้นภายใต้การใช้งานจริงเป็นวิธียับยั้ง Failure รวมถึงเพิ่มประสิทธิภาพของระบบวิธีหนึ่ง เนื่องจากบ่อยครั้ง Failure ที่เกิดขึ้นสามารถตรวจสอบได้จาก Event log ซึ่งเป็นตัวแทนในการวิเคราะห์และสื่อถึงพฤติกรรมที่ปกติและไม่ปกติที่เกิดขึ้นระหว่างการทำงาน เพราะเมื่อมีสิ่งผิดปกติ log สามารถนำมาวิเคราะห์ Service Failure ท่ามกลางอุปกรณ์ต่างๆที่มีอยู่ในระบบ โดยการตรวจสอบความน่าเชื่อถือในลักษณะนี้เราเรียกว่า log-based failure analysis หรือการวิเคราะห์ Failure จากข้อมูล Log ซึ่งได้มีงานวิจัยจำนวนมากที่ได้กล่าวถึงการพัฒนาและเพิ่มประสิทธิภาพในการวิเคราะห์ Failure ยกตัวอย่างเช่น การตรวจสอบในระบบโดยตรงจาก log ที่เกิดขึ้น [8] แต่ข้อเสียของการวิเคราะห์ในลักษณะนี้ คือ เมื่อเกิดปัญหากับระบบการให้บริการ ผู้ดูแลจะต้องรับมือและจัดการกับปัญหาเองทุกครั้ง [9] กล่าวถึงวิธีในการลดจำนวนข้อมูล Log ที่ไม่จำเป็นหรือไม่เกี่ยวข้องในการวิเคราะห์ แต่ไม่ได้มีการ Correlate เพื่อวิเคราะห์สิ่งที่เกิดขึ้นจริงๆ และการลดจำนวน log ที่ไม่จำเป็นนั้น อาจส่งผลให้เกิดความผิดพลาดในการวิเคราะห์ Failure ก็เป็นได้ [10] เป็นการวิเคราะห์ Failure เพื่อหาต้นตอของสาเหตุและระบุปัญหาที่เกิดขึ้น หรือ Root Cause Analysis ซึ่งเป็นแนวทางในการแก้ปัญหาและพัฒนาระบบให้มีประสิทธิภาพอีกวิธีหนึ่ง [5, 11] ใช้วิธี rule-based และ case-based แบบอัตโนมัติเพื่อเพิ่มประสิทธิภาพในการทำงาน แต่อย่างไรก็ตามจะเห็นว่าในงานวิจัยนี้ไม่ได้คำนึงถึงข้อมูลที่มีขนาดใหญ่ในเชิง Big data [12, 13] ใช้การเพิ่ม Logging rule เพื่อตรวจสอบ Availability และ Reliability ของระบบ แต่ในการพัฒนาการผลิต log นั้น จำเป็นที่จะต้องแก้ไข code ของโปรแกรมภายในเพื่อกำหนดให้เป็นไปตาม logging rule ที่ต้องการ ซึ่งหากโปรแกรมเป็นแบบ Package จะไม่สามารถแก้ไข code ได้ หรือหากสามารถแก้ไขได้ก็อาจส่งผลถึงการขยายงานในอนาคตและส่งผลถึงเรื่อง Compatibility อีกด้วย นอกจากนี้แล้ว ในการตรวจสอบ Service failure ยังรวมถึงเรื่องของ Online failure prediction ซึ่ง [14, 15] นำเสนอเรื่อง Proactive self-adaptation เพื่อตรวจจับพฤติกรรมที่เปลี่ยนแปลงไป โดยจะเป็นลักษณะของการประเมินความเสี่ยงที่อาจเกิดขึ้นกับระบบ แต่ไม่ได้หาต้นตอของปัญหาที่เกิดขึ้นกับระบบที่แท้จริง

จากปัญหาที่กล่าวมาแล้วข้างต้น งานวิจัยนี้จึงได้ทำการพัฒนาและแก้ไขปัญหาที่เกี่ยวข้องกับระบบงาน Monitoring เพื่อรองรับการทำงานแบบ Real-time โดยนำสถาปัตยกรรมเชิง Big data มาใช้ในการวิเคราะห์ข้อมูล Log ในลักษณะที่เป็น Raw log เพื่อไม่ให้เกิดปัญหาเกี่ยวกับการแก้ไข Code รวมถึงรับมือกับปัญหาเรื่องความซับซ้อนของระบบกับขนาดของข้อมูล Log และมุ่งเน้นไปที่การคำนวณหาความน่าจะเป็นในการกำหนดระยะเวลา (Deadline) เพื่อให้เป็นไปตาม SLA

deadline สำหรับตรวจสอบระบบ Real-time system ที่มีความต้องการเป็นแบบ Soft deadline โดยประยุกต์ทฤษฎีทางสถิติเพื่อประเมินถึงโอกาสที่จะไม่เป็นไปตามข้อตกลงในส่วนของ Deadline Constraint ในระบบ นอกจากนี้ งานวิจัยยังนำเสนอเรื่องการหาความสัมพันธ์ของ Event log ที่เรียกว่า Log correlation โดยตรวจสอบและวิเคราะห์ Failure แบบเรียลไทม์เพื่อหาต้นตอของสาเหตุและระบุปัญหาที่เกิดขึ้นจากข้อมูลล็อกในแต่ละ Resource เพื่อตรวจสอบว่าขณะนี้ Service ที่ให้บริการอยู่นั้นยังมีสภาพพร้อมใช้งานอยู่หรือไม่ กล่าวคือ การให้บริการของ Service ยังคงสามารถให้บริการได้อย่างถูกต้องนั้นเอง

1.2 วัตถุประสงค์

โครงงานวิจัยนี้มีวัตถุประสงค์ ดังนี้

1. เพื่อพัฒนาวิธีการตรวจสอบความล้มเหลวในระบบและหาความสัมพันธ์ของ Event log correlation โดยใช้วิธี Log-based failure analysis แบบ Soft Real-time และนำไปประยุกต์ใช้งานและรับมือกับปริมาณข้อมูล log ที่เกิดขึ้นจริง
2. เพื่อพัฒนาโครงสร้างสำหรับรองรับข้อมูลที่มีลักษณะเป็น Data stream และตรวจสอบให้ระบบมีประสิทธิภาพตามระดับข้อตกลงของการให้บริการ (SLA) โดยนำทฤษฎีทางด้านสถิติมาคำนวณความน่าจะเป็นในการละเมิด Deadline constraint

1.3 ขอบเขตของการวิจัย

เครื่องมือที่ใช้ในงานวิจัยนี้ประกอบด้วย

- Hadoop version 0.20.2-cdh3u5
- Hadoop-0.20-fuse
- Flume version 0.9.4-cdh3u5
- SEC (Simple Event Correlator) version 2.5.3
- java version 1.6.0_24
- Service ตัวอย่างที่ใช้ในการตรวจสอบคือ Web Application ของคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้รับในงานวิจัยนี้ ได้แก่

1. เข้าใจหลักการในการวิเคราะห์ Failure ในระบบ ด้วยวิธี Rule-based approach
2. ได้ความรู้เกี่ยวกับรูปแบบการหาความสัมพันธ์แบบ Event correlation และการทำงานของ HDFS

3. ได้ต้นแบบของ Framework ในการจัดการกับข้อมูล Log ในโครงสร้างการทำงานเชิง Big data เพื่อสามารถรองรับขนาดของข้อมูลได้ง่ายขึ้น
4. สามารถนำความรู้จากผลการวิจัยนี้มาประยุกค์ใช้คำนวณหาความน่าจะเป็นในการกำหนด Deadline ของแต่ละระบบ และคำนวณหาโอกาสที่ระบบ Monitoring จะ Detect เกิน Deadline ที่กำหนด

1.5 แผนการดำเนินการวิจัย

ในขั้นตอนการดำเนินงานแบ่งออกเป็น 5 ขั้นตอน ดังนี้

1.5.1 ศึกษางานวิจัยที่เกี่ยวข้อง

ทำการศึกษางานวิจัยที่เกี่ยวข้อง เช่น ศึกษาเกี่ยวกับการประยุกต์ทฤษฎี Central limit theorem, Soft real-time, Normal distribution รวมถึงศึกษาเกี่ยวกับการวิเคราะห์ Failure ของระบบที่มีอยู่ในปัจจุบัน และรูปแบบความสัมพันธ์ของ Event log ในแบบต่างๆ ศึกษาเครื่องมือที่ใช้สำหรับการ monitoring เพื่อใช้ควบคู่กับการทำงานของระบบแฟ้มแบบกระจาย อย่างเช่น HDFS เป็นต้น

1.5.2 ศึกษาเครื่องมือที่ใช้ในงานวิจัย

ทำการศึกษาวิธีการใช้เครื่องมือต่างๆที่จะนำมาใช้ในการสร้างระบบ เช่น ศึกษาการทำงานของ Flume ในการจัดเก็บข้อมูล Log, ทำการทดลอง Simple Event Correlator (SEC) ในการ Correlation Event ต่างๆที่เกิดขึ้น ทกลอง Flume ในการเก็บข้อมูล log จากอุปกรณ์ต่างๆเพื่อส่งต่อไปเก็บยัง Storage บน HDFS เป็นต้น

1.5.3 ขั้นตอนออกแบบและทำการสร้าง Framework การทำงานของระบบ

ทำการออกแบบและสร้าง Framework การทำงานของระบบเพื่อรองรับ Soft real-time monitoring ภายใต้เงื่อนไขของข้อจำกัดทางเวลา (Timing Constraint)

1.5.4 ขั้นตอนการทดลองเพื่อทดสอบความสามารถ

นำ Framework การทำงานไปทำการทดลองใช้จริง เพื่อทดสอบความสามารถและประสิทธิภาพที่ระบบสามารถรองรับได้ โดยในส่วนของการทดสอบความสามารถนั้นคือการตรวจสอบการทำงานของ Framework ที่สร้างขึ้นว่าสามารถทำงานได้ตามที่ได้มีการออกแบบมาหรือไม่และเกิดสิ่งผิดปกติภายใต้สภาวะการใช้งานจริงหรือไม่ รวมถึงคำนวณความน่าจะเป็นที่ระบบจะสามารถตรวจสอบได้ภายใต้เงื่อนไขที่กำหนด Deadline ในแต่ละช่วงเวลา

1.5.5 ขั้นตอนการสรุปผลการทดลองและจัดทำวิทยานิพนธ์

สรุปผลการทดลองโดยนำผลการทดลองที่ได้มาสรุปว่าหา Framework ที่ได้ทำการสร้างขึ้นมานั้นมีความสามารถและประสิทธิภาพเป็นที่น่าพอใจหรือไม่ พร้อมทั้งหาเหตุผลมาอธิบายถึงผลที่เกิดขึ้น

1.6 ผลงานตีพิมพ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้นำเสนอในการประชุมวิชาการ ดังนี้

- Panya Kittipipattanathaworn and Natawut Nupairoj , SLA Guarantee Real-Time Monitoring System with Soft Deadline Constraint , The 11th International Joint Conference on Computer Science and Software Engineering (JCSSE 2014) , Pattaya, Thailand, May 14-16, 2014

บทที่ 2
บทนำ

วิธีการตรวจสอบและวิเคราะห์ข้อมูล Log ที่มีลักษณะเป็น data stream และการประยุกต์ทฤษฎีทางด้านสถิติเพื่อคำนวณความน่าจะเป็นในการละเมิด SLA timing constraint เพื่อให้ระบบมีประสิทธิภาพตามระดับข้อตกลงของการให้บริการ (SLA) ในระบบที่มีลักษณะเป็น Soft real-time จะมีทฤษฎีที่เกี่ยวข้องคือ

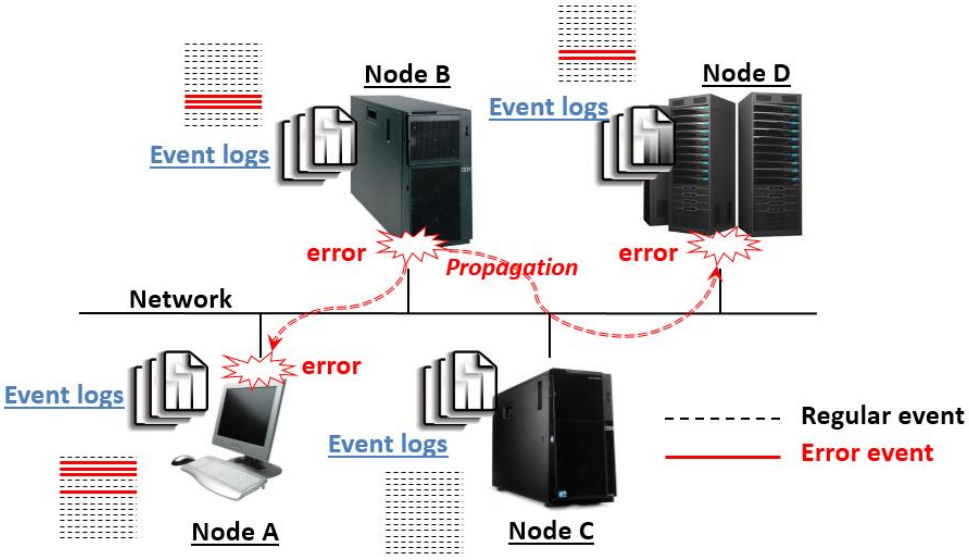
2.1 Event log

Event log คือ ข้อมูล log ที่ถูกสร้างขึ้นระหว่างการให้บริการของระบบ เพื่อสื่อถึงพฤติกรรมต่างๆที่เกิดขึ้นทั้งเหตุการณ์ปกติและ ไม่ปกติในส่วนการทำงานของอุปกรณ์หรือ Service นั้นๆ ซึ่งปกติแล้วการสร้างข้อมูล Log ถูกใช้งานอย่างกว้างขวางในหลายๆ Application รวมทั้ง Operating system, Control system, Mobile device, Supercomputer, Application component ดังนั้นเมื่อมีสิ่งผิดปกติเกิดขึ้น ข้อมูล log สามารถเป็นตัวแทนที่สื่อถึงพฤติกรรมของระบบ และนำไปวิเคราะห์ failure ท่ามกลางอุปกรณ์ต่างๆที่มีอยู่ในระบบได้

```
Dec 11 21:52:22 panya crontab[2270]: (user1) END EDIT (user1)
Dec 11 22:00:01 panya CRON[30567]: (user1) CMD (/home/user1/pingtest.sh > /dev/null)
Dec 11 22:00:47 panya postfix/pickup[27594]: 2DBB518816FO: uid=1002 from<user1>
Dec 11 22:00:47 panya postfix/cleaning[30664]: 2DBB518816FO: message-id=<20121211600047.2DBB518816FO@panya>
Dec 11 22:00:47 panya postfix/qmgr[1788]: 2DBB518816FO: from=<user1@panya>, size=1643, nrcpt=1 (queue active)
Dec 11 22:00:47 panya post/local[30666]: 2DBB518816FO: to=<user1@panya>, orig_to=<user1>, relay=local, delay=0.21, delays=0.14/0/0/0.06,
dsn=2.0.0, status=sent (delivered to mailbox)
Dec 11 22:00:47 panya postfix/qmgr[1788]: 2DBB518816FO: removed
```

ภาพที่ 2.1 แสดงตัวอย่างของ Event log

ภาพที่ 2.1 เป็นตัวอย่างของ log ซึ่งปกติแล้ว ภายในจะประกอบด้วยข้อมูลต่างๆ เช่น Timestamp เป็นส่วนบอกถึงเวลาที่ Event เกิดขึ้น, ข้อมูล IP address หรือชื่อของ node ที่ generate log และ Text message เป็นข้อมูลเกี่ยวกับ event ที่บ่งบอกถึงเหตุการณ์ปกติ (regular) ไปจนถึง Error ที่เกิดขึ้นในช่วงที่ระบบมีการทำงาน [16] โดยที่ regular event บอกถึงพฤติกรรมที่ปกติไม่มีการ Error เช่น Disk mount, network status, user connection ส่วน Error event จะบอกเกี่ยวกับปัญหาของ Hardware, Software หรือการทำงานที่ผิดพลาดและไม่สำเร็จ รวมถึงระดับความรุนแรงของ event เป็นต้น

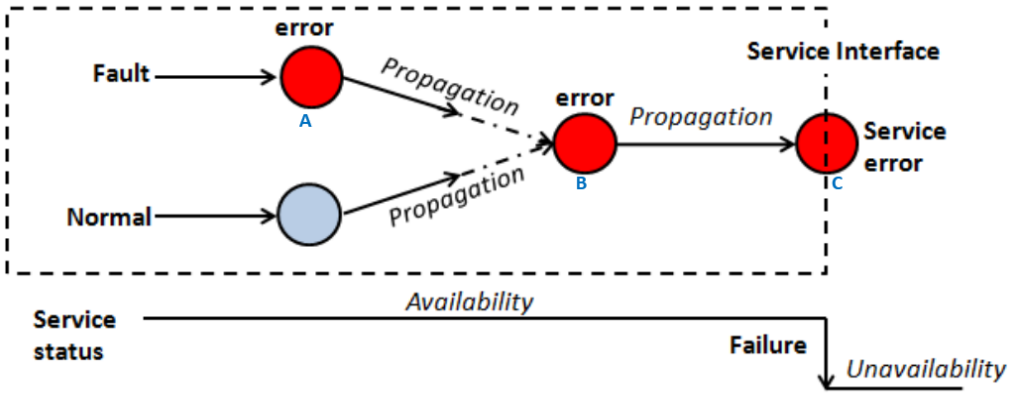


ภาพที่ 2.2 แสดงลักษณะการทำงานในลักษณะขึ้นต่อกันและการแพร่กระจายของ Error

จากภาพที่ 2.2 เป็นระบบการให้บริการในลักษณะขึ้นต่อกัน [17] แบบ Distributed ที่ประกอบด้วยอุปกรณ์หลาย node ทำงานร่วมกันผ่าน Network ซึ่งจะเห็นว่า Node แต่ละ Node มีการผลิต Event log เพื่อบอกถึงการทำงานในส่วนของตน แต่เมื่อเกิด error event ขึ้นที่ Node หรืออุปกรณ์ตัวใดตัวหนึ่ง อาจมีการแพร่กระจายของ Error (Error propagation) ซึ่งอาจส่งผลกระทบต่อ Node ในแต่ละ Node ที่มีลักษณะการทำงานที่ขึ้นต่อกัน ทำให้ยากที่จะ Monitor เพื่อระบุถึงจุดตั้งต้นของปัญหาที่เป็นสาเหตุทำให้การให้บริการของ Service หรือ Subservice อื่นๆไม่สามารถใช้งานได้

2.1.1 Fault, Error, Failure

ความน่าเชื่อถือของระบบจะมองในมุมของการให้บริการและพฤติกรรมของระบบที่ผู้ใช้งานได้รับ ซึ่งผลลัพธ์ของการให้บริการที่ Service จะได้รับ ในที่นี้เรียกว่า Service interface จากภาพที่ 2.3 ผู้ใช้งานจะเห็น Service Interface ในจุด C ไม่สามารถใช้งานได้ แต่เมื่อมองให้ลึกถึงภายในการทำงานของระบบ สาเหตุที่เป็นต้นเหตุทำให้ Service ไม่สามารถให้บริการได้ คือ Fault ที่ส่งผลให้เกิด Error ขึ้นกับ Node A และแพร่กระจาย Error ต่อไปยัง Node B ซึ่งเป็นส่วนที่เกี่ยวข้องในการทำงานของระบบทำให้เกิด Error จนท้ายที่สุดแล้วทำให้ Service ไม่สามารถให้บริการได้ถูกต้องหรือเกิด Failure ขึ้นกับ Service ดังนั้นส่วนที่จะต้องซ่อมแซมคือ Node A เนื่องจากเป็นต้นตอของปัญหาที่ทำให้เกิด Failure ใน Service ทั้งหมด



ภาพที่ 2.3 แสดงตัวอย่างการเกิด Failure ของระบบ (adapted from [4])

สาเหตุที่ทำให้สถานะของการให้บริการเปลี่ยนแปลงจาก Availability อาจเกิดได้จากหลายกรณี เช่น เกิด failure เนื่องจาก Function การทำงานไม่เป็นไปตามที่ควรจะเป็น ซึ่งการที่จะเปลี่ยนสถานะเพื่อกลับไปเป็น Availability ได้นั้น จะต้องมีการฟื้นฟูหรือซ่อมแซม Service เสียก่อน ซึ่งตัว Error จะเป็นส่วนของระบบที่เป็นต้นเหตุทำให้เกิด failure ต่อมาเรื่อยๆ ส่วน Fault คือสาเหตุของการตัดสินหรือสมมติฐานสาเหตุของ Error ซึ่ง Fault สามารถเกิดได้จากส่วนของการพัฒนาระบบ (Development fault), ผลกระทบจากความผิดพลาดของ Hardware ต่างๆ (Physical fault) รวมถึง fault ที่เกิดจากภายนอก (External fault)

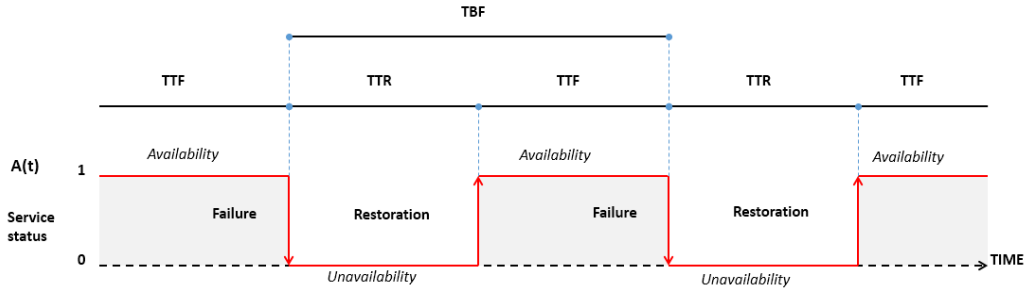
โดยความสัมพันธ์ของ Fault, Error และ Failure มีความสัมพันธ์กันอย่างที่แสดงในรูปที่ 3 ซึ่ง fault สามารถเกิดขึ้นได้ทั้งภายในและภายนอก จากนั้นอาจทำให้เกิด Failure โดยอาจมีการแพร่กระจายของ Error (error propagation) ไปยัง service interface ทำให้เกิดความล้มเหลวต่อการให้บริการ หรืออาจไม่ส่งผลกระทบต่อการให้บริการเลยก็เป็นได้ในกรณีที่มี Redundant

2.2 Service Level Agreement (SLA)

ในงานวิจัยนี้ลักษณะการทำงานของระบบ IT จำเป็นจะต้องมีการการันตีคุณภาพการให้บริการ (Quality of Service) เป็นส่วนสำคัญ ดังนั้นปัญหาหลักสำหรับผู้ให้บริการคือทำอย่างไรให้ผู้ใช้บริการมั่นใจได้ว่าการให้บริการนั้นจะเป็นไปตาม SLA หรือตามข้อตกลงของระดับการให้บริการที่ได้ให้ไว้ ซึ่งบ่อยครั้งที่การตรวจสอบ Service Availability รวมถึง MTBF และ MTTR ได้ถูกเลือกเป็นส่วนหนึ่งในการตรวจสอบ SLA [5] ดังนั้น ผู้ให้บริการจะต้องมีประสิทธิภาพและความสามารถในการตรวจสอบ Failure หรือ Detect สิ่งผิดปกติที่เกิดขึ้นในระบบของการให้บริการอย่างแม่นยำและรวดเร็ว โดยการเพิ่มประสิทธิภาพความแม่นยำรวดเร็วนั้นจำเป็นต้องมองในมุมมองของโครงสร้างภายในระบบ กล่าวคือ จะต้องรู้ถึงรายละเอียดโครงสร้างที่มีลักษณะขึ้นต่อกันของ Service, Subservice และ Resource ตลอดจน SLA ของผู้ใช้บริการ QoS parameter และการใช้บริการในเวลานั้นๆ

2.2.1 ความพร้อมใช้งาน (Availability)

Availability คือ ความสามารถที่ระบบสามารถใช้งานได้ โดยในที่นี้หมายถึง Service availability ที่เป็นระยะเวลาพร้อมใช้งานเมื่อผู้ใช้ต้องการใช้ ซึ่งจะเกิดจากการทำงานของ subservice และ Resource ต่างๆทำงานร่วมกัน เช่น Web application service เกิดจากการทำงานร่วมกันของ DNS Service, Authentication service, Storage service เป็น Subservice โดย Service availability สามารถหาได้จากอัตราส่วนของเวลาที่สามารถใช้งาน service ได้ต่อเวลารวมทั้งหมด



ภาพที่ 2.4 แสดงการคำนวณสถานะของการให้บริการ

$$A = \frac{MTTF}{MTTR + MTTF} = \frac{MTTF}{MTBF} = \frac{Service\ Availability}{Total\ time} \quad (2.1)$$

โดยที่ ;

MTTF (Mean Time To Failure) คือ เวลาเฉลี่ยก่อนการเกิด Failure

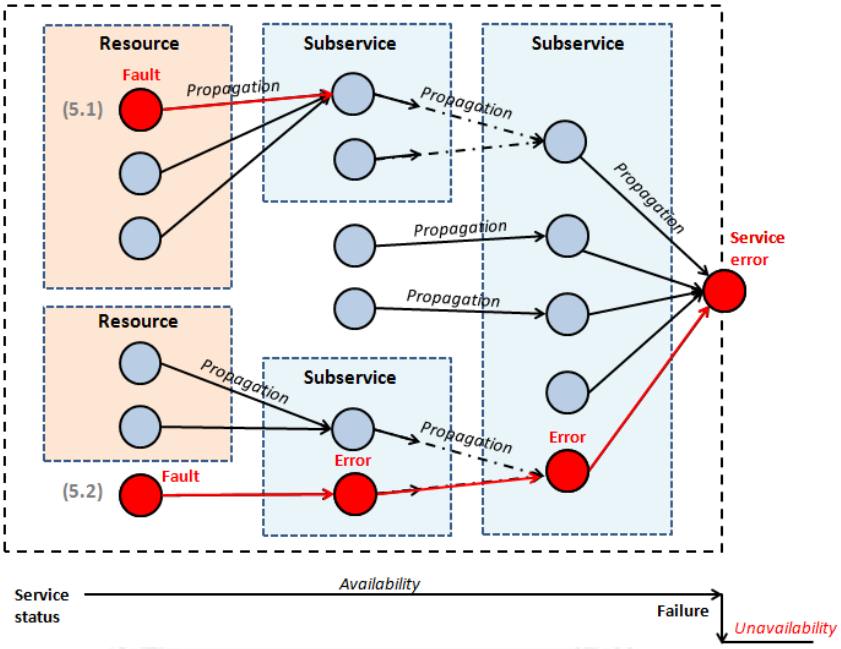
MTTR (Mean Time To Repair) คือ เวลาเฉลี่ยในการซ่อมแซมระบบ ยิ่งเวลาเฉลี่ยในการซ่อมแซมน้อยจะยิ่งเป็นผลดี รวมถึงการตรวจสอบสิ่งผิดปกติในระบบหรือการ Detect จะต้องมีลักษณะที่เป็น Real-time กล่าวคือ ยิ่ง Detect ได้เร็วก็จะยิ่งมีเวลาเฉลี่ยในการซ่อมแซมน้อยเช่นกัน

MTBF (Mean Time between Failure) = (MTTR+MTTF) คือ เวลาเฉลี่ยระหว่างการเสียหาย ดังนั้นถ้าเกิด Time to repair ของระบบน้อยแสดงว่ามี Service availability ที่สูง

Service Availability คือ ช่วงเวลาที่ระบบสามารถใช้งานได้ หมายถึงช่วงเวลาที่ user สามารถเข้าใช้งาน Web application ที่ต้องการจากระบบได้ ถึงแม้ว่าบางครั้ง Resource บางตัวอาจจะไม่สามารถใช้งานได้แต่ Service ยังคงสามารถให้บริการต่อไปได้

Total time คือเวลารวมทั้งหมดของระบบ นั้นคือ Service availability + Service unavailability โดยที่ Service unavailability คือช่วงเวลาที่ user ไม่สามารถเข้าใช้งาน Web application ได้ ซึ่งอาจเกิดได้เนื่องจากช่วงเวลานั้นมี Service หรือ Subservice บางตัวอาจจะมีปัญหาทำให้ไม่สามารถใช้งานระบบได้ เช่นการทำงานของ DNS ผิดพลาดทำให้ระบบไม่สามารถใช้งานได้ หรือ Database มีปัญหาเป็นต้น

โดยการตรวจสอบ Availability นั้น เราสามารถตรวจสอบได้จาก log ของแต่ละ Resource, Subservice และ Service ว่ายังคงมีการสร้าง log ในลักษณะที่เป็นปกติออกมาหรือไม่ จากรูปที่ 5.1 หากมี Fault บางอย่างเกิดขึ้นใน Resource บางตัว แต่ในการทำงานของระบบมีลักษณะเป็น redundant จึงไม่ส่งผลกระทบต่อ Availability ของระบบ ในขณะที่รูปที่ 2.5 เกิด Fault ซึ่งส่งผลให้ service อื่นๆที่มีลักษณะการทำงานแบบขึ้นต่อกันเกิด Error และเป็นผลให้สถานะของ Service ใช้งานไม่ได้ไปด้วย เราจะเรียกสถานะนี้ว่าเป็น Unavailability



ภาพที่ 2.5 แสดงตัวอย่างการเกิด Failure ของระบบ

หรือหากโดยปกติระบบมีการส่ง Log ออกมาตลอดเวลา แต่หากมีช่วงเวลาที่ log ไม่ถูกสร้างขึ้นเมื่อถูกการใช้งานหรือเกิด Error ขึ้น อาจสันนิษฐานได้ว่าระบบนั้นๆอาจจะมีปัญหาเกิดขึ้น หรือกรณีที่ Service ไม่สามารถใช้งานได้โดยเกิด Error อาจจะต้องทำการวิเคราะห์หาต้นตอของปัญหาว่าเกิดจากสาเหตุอะไร โดยสามารถทำได้โดยการวิเคราะห์ log ดังนั้นสถานะของ Service availability จะมีอยู่ 2 สถานะคือ Availability กับ Unavailability

$$A(T) = \begin{cases} 1 & Availability \\ 0 & Unavailability \end{cases} \tag{2.2}$$

2.2.2 เวลาในการตอบสนอง (Response time)

Response time คือ ระยะเวลาที่ใช้ในการส่งคำสั่งการร้องขอ Request บางอย่างเพื่อให้ได้ผลลัพธ์ที่ต้องการ นั่นคือช่วงเวลาที่ user เริ่มส่งงานที่ต้องการไปยังระบบ จากนั้นระบบจะมีการ Execute process จนกระทั่งระบบส่งผลลัพธ์กลับมาแสดงผล ยกตัวอย่างเช่น ช่วงเวลาที่ user ทำการ Access ไปยังหน้า web page นั้น response time ก็คือช่วงเวลาที่ user ต้องรอจนกระทั่งเห็นหน้า web page นั้นๆแสดงผลลัพธ์ออกมายังหน้าจอ

2.2.3 เวลาในการตรวจจับ (Detection time)

Detection time คือระยะเวลาที่ระบบ Monitoring สามารถตรวจจับความผิดปกติของข้อมูล log ได้ โดยหากระบบ Monitoring สามารถ Detect สิ่งผิดปกติได้เร็วก็จะช่วยให้ระบบมี Service Availability ที่สูงขึ้น

2.3 Event Correlation

Event Correlation [18] คือเทคนิคที่ใช้เพื่อลดจำนวนของ Event ที่มีจำนวนมากๆและแยกแยะ Event ที่มีความสำคัญจริงๆในข้อมูลทั้งหมด ในลักษณะที่เชื่อมโยงกัน ซึ่งขอบเขตเหล่านี้จะรวมถึง Network management, System management และ Service-Level Management ด้วย

Event และ Event Correlator

Event Correlator โดยทั่วไปจะอยู่ในส่วนการทำงานของ Management platform ตั้งแต่หนึ่งอุปกรณ์ไปจนถึงระดับ platform หรือที่เราเรียกว่า Network management System โดยทำการหาความสัมพันธ์เฉพาะส่วนการทำงานที่ผู้ดูแลสนใจ เช่น ระดับการให้บริการไม่ได้เป็นไปตามที่ให้ไว้กับผู้ใช้บริการ, มีการ Reboot device, หรือ CPU ของ server มีการใช้งานเต็ม 100% มานานเกิน 15 นาที เป็นต้น

Event Correlator มีบทบาทสำคัญในการผสานความสัมพันธ์ของการจัดการเรื่อง Network, system และ Service event รวมเข้าด้วยกัน เพื่อทำการหาความสัมพันธ์เกี่ยวกับสิ่งที่เกิดขึ้น ยกตัวอย่างเช่น อะไรที่เป็นสาเหตุ Failure ที่ถูกทำให้เกิดความล้มเหลวในโครงสร้างการให้บริการของระบบ และปัญหาจริงๆของการ Failure เกิดจากส่วนใด เนื่องจากการทำงานในอุปกรณ์หลายๆส่วนอาจเกิดปัญหาขึ้น ในขณะที่ต้นตอของปัญหาอาจมีเพียงปัญหาเดียว

Event correlation สามารถแบ่งขั้นตอนเป็น ได้ดังนี้

- **Event filtering**: ทำการแยกเหตุการณ์ที่ไม่เกี่ยวข้องและไม่น่าสนใจออก เช่น คัดกรองข้อมูลที่น่าสนใจเฉพาะข้อมูลที่เกี่ยวข้องกับการเกิด Fault ของ Resource เป็นต้น
- **Event Aggregation**: ลักษณะคือการรวม Event ที่เหมือนกันเพื่อลดการซ้ำซ้อนของข้อมูล เนื่องจากอาจมีการส่ง Event ที่เหมือนกันเป็นจำนวนมาก

- **Root cause analysis**: เป็นกระบวนการสุดท้ายของ event correlation โดยการวิเคราะห์จะขึ้นอยู่กับ Event และโครงสร้างของสภาพแวดล้อม เพื่อตรวจจับบาง event ที่อาจนำมาอธิบายได้ เช่น กรณีของ Database D รันอยู่บน Server S และ server เกิด overload (มีการใช้ CPU 100% นานซักระยะหนึ่ง) อาจทำการสร้าง Event เป็น “Server S is durably overloaded” ซึ่งเป็นต้นเหตุของปัญหา ที่สามารถตอบปัญหาของการละเมิด SLA ตามระดับการให้บริการได้

2.4 Big Data

Big data หรือฐานข้อมูลขนาดใหญ่ ในที่นี้หมายถึง เซตของข้อมูลที่ใหญ่มากขึ้นเรื่อยๆและเกิดขึ้นอย่างรวดเร็ว จนยากที่จะใช้เครื่องมือการบริหารที่ทำงานด้วยมือคนแบบที่เคยเป็นมา รวมถึงสภาพแวดล้อมในระบบที่มีความแตกต่างกันและเปลี่ยนแปลงได้ตลอดเวลา จนทำให้เกิดความซับซ้อนมากขึ้นเรื่อยๆภายในระบบ [19] โดยขนาดและความซับซ้อนของข้อมูลนี้รวมถึง การจัดเก็บ การค้นหา การวิเคราะห์ และแนวโน้มของชุดข้อมูลต่างๆ ซึ่งปัญหาในการทำงานกับสภาพแวดล้อมที่มีลักษณะเชิง Big data คือ ปริมาณ (Volume), ความเร็ว (Velocity), และความหลากหลาย (Variety) [20]

- **ปริมาณ (Volume):** องค์กรต่างๆจะมีการสร้างข้อมูลต่างๆรวมถึงข้อมูล Log ที่เพิ่มมากขึ้นเรื่อยๆ จนถึงขนาดเทราไบต์
- **ความเร็ว (Velocity):** การที่ข้อมูลมีการไหลเข้ามาตลอดเวลา จำเป็นจะต้องใช้ประโยชน์จากข้อมูลเหล่านั้นให้มากที่สุดในลักษณะเรียลไทม์ เพื่อวิเคราะห์ข้อมูลได้ถูกต้องและรวดเร็วยิ่งขึ้น
- **ความหลากหลาย (Variety):** อาจเป็นข้อมูลชนิดใดก็ได้ ไม่ว่าจะมีโครงสร้างหรือไม่มีโครงสร้างก็ตาม เช่น ข้อความ, ข้อมูลจากเซ็นเซอร์, เสียง, วิดีโอ, ข้อมูลจากการคลิก, ข้อมูลล็อกและข้อมูลอื่น ๆ

2.5 Real-time Monitoring System

องค์ประกอบสำคัญของระบบ Real-time monitoring system คือความสามารถที่ระบบจะตรวจสอบความผิดพลาดได้ภายในระยะเวลาที่กำหนด (Deadline) ตาม SLA ซึ่งสามารถแบ่งระดับในการตรวจสอบได้ 2 ระดับ คือ [21]

Soft: ผลลัพธ์ที่ได้รับหลังจากระยะเวลาที่กำหนด (Deadline) ยังคงสามารถนำมาใช้งานได้ แต่ส่งผลให้ละเมิดข้อตกลงในการให้บริการ

Hard: ไม่สามารถละเมิดข้อตกลงของการให้บริการโดยเด็ดขาด

2.6 การกระจายของค่าความน่าจะเป็น (Probability Distribution)

2.6.1 ความน่าจะเป็น (Probability) [22]

ความน่าจะเป็น (Probability) ในที่นี้หมายถึง สถิติในการคาดการเหตุการณ์ที่จะเกิดขึ้นในอนาคตหรือสิ่งที่จะเกิดขึ้นกับประชากรส่วนมาก โดยอาศัยข้อมูลที่เกิดขึ้นมาแล้วและเป็นเพียงตัวอย่างส่วนหนึ่งที่ถูกดึงออกมาจากประชากรส่วนมากมาเป็นพื้นฐาน ดังนั้นเราจึงจำเป็นต้องรู้ว่าเหตุการณ์แต่ละเหตุการณ์มีโอกาสเกิดขึ้นได้เท่าไหร่ ซึ่งเป็นคุณสมบัติตามธรรมชาติของสิ่งที่เรากำลังสนใจ เพราะถ้าเราไม่สามารถรู้ว่าเหตุการณ์ที่เราสนใจมีโอกาสเกิดขึ้นมากหรือน้อยเท่าไหร่นั้น จะทำให้ไม่สามารถคาดการณ์อนาคตหรือคาดการณ์ประชากรได้ถูกต้อง นั่นแปลว่าเราก็ไม่สามารถใช้การคำนวนสถิติเพื่อการนี้ได้ ดังนั้น ข้อมูลแต่ละประเภทก็จะมีลักษณะค่าโอกาสที่แตกต่างกัน และเมื่อเรานำค่าโอกาสมาแสดงในรูปกราฟการกระจาย (Distribution) จะเรียกว่าการกระจายของค่าโอกาส (Probability Distribution)

2.6.2 ตัวแปรสุ่ม (Random Variable)

การสุ่ม (Randomization) ถือว่าเป็นสิ่งจำเป็นสำหรับเรื่อง Probability เนื่องจากเครื่องมือทางสถิติจะหมดความหมายทันทีถ้าหากว่าการทำการทดลองใดๆ เพื่อให้ได้ข้อมูลมา ไม่ได้เกิดจากการสุ่มที่ถูกต้อง นั่นแปลว่า โอกาสของเหตุการณ์ต่างๆที่ได้จากการทดลองไม่ได้เป็นไปตามธรรมชาติที่ควรจะเป็น

2.6.3 การแจกแจงแบบโค้งปกติ (Normal distribution)

การแจกแจงแบบโค้งปกติ เป็นการแจกแจงของข้อมูลที่ได้จากตัวแปร ที่มีลักษณะต่อเนื่อง (Continuous variable) โดยมีคุณสมบัติของโค้งปกติ ดังนี้

1. พื้นที่หรือความน่าจะเป็น (Probability) ภายใต้โค้งปกติมีค่าเท่ากับ 1
2. ความสูงของโค้งที่สูงที่สุดอยู่ที่ค่า μ
3. โค้งมีลักษณะเป็นรูประฆังคว่ำ สมมาตร และมีค่าเฉลี่ย มัธยฐาน และฐานนิยมเท่ากัน
4. ลักษณะการกระจายภายใต้โค้งปกติมีลักษณะที่ว่าในช่วงบวกลบ 1 เท่าของค่าเบี่ยงเบนมาตรฐานจากค่าเฉลี่ยหรือจุดกลางของโค้ง (μ±1) มีพื้นที่ประมาณ 68% และ (μ±2) มีพื้นที่ประมาณ 95% และ (μ±3) มีพื้นที่ประมาณ 99%

2.6.4 ค่าเบี่ยงเบนมาตรฐาน (Standard deviation)

ค่าเบี่ยงเบนมาตรฐานใช้วัดการกระจายของข้อมูล เพื่อพิจารณาว่าคะแนนแต่ละตัวจะแตกต่างไปจากค่ากลางมากน้อยเพียงใด คำนวณโดยเอาคะแนน X แต่ละตัวลบด้วยค่าเฉลี่ย ($\overline{X}$) ของข้อมูลชุดนั้น ซึ่ง $X - \overline{X}$ แต่ละตัวอาจมีค่าเป็นลบ ($X < \overline{X}$) หรือบวก ($X > \overline{X}$) จึงต้องยกกำลังสองของคะแนนเบี่ยงเบนแต่ละตัวนั้นเพื่อให้เครื่องหมายหมดไป แล้วหาค่าเฉลี่ยของผลบวกของกำลังสองของคะแนนเบี่ยงเบน คือ ซึ่งจะได้รับค่าความแปรปรวน ถ้าถอดรากที่สองของค่าความแปรปรวนจะได้ค่าความเบี่ยงเบนมาตรฐาน

ความแปรปรวน (Variance) คือ ค่าเฉลี่ยของผลรวมทั้งหมดของคะแนนเบี่ยงเบนยกกำลังสอง ใช้สัญลักษณ์ S2 แทนความแปรปรวนของกลุ่มตัวอย่างและ s 2 แทนความแปรปรวนของประชากรซึ่งหาได้จากสูตร

ความแปรปรวนประชากร $$S^2 = \frac{1}{N}\sum_{i=1}^{n}(X_i - \overline{X})^2 \quad (2.3)$$

- $\overline{X}$ คือ ค่าเฉลี่ยของกลุ่มตัวอย่าง (Sample mean) เขียนแทนด้วย Mean ($\overline{X}$)

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{n} X_i \quad (2.4)$$

- S ส่วนเบี่ยงเบนมาตรฐาน (Standard Deviation) คือ รากที่สองของความแปรปรวน ส่วนเบี่ยงเบนมาตรฐานของประชากร S ใช้สูตร

$$S = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(X_i - \overline{X})^2} \quad (2.5)$$

2.6.5 คะแนนมาตรฐาน Z

ในการที่จะนำคะแนนแต่ละกลุ่มมาเปรียบเทียบกันไม่สามารถทำได้ ดังนั้นในการที่จะนำคะแนนมาเปรียบเทียบกันให้ได้ความหมายนั้น ทำได้โดยเปลี่ยนคะแนนดิบให้เป็นคะแนนมาตรฐาน ซึ่งคะแนนมาตรฐานแต่ละชนิดจะมีค่าเฉลี่ย และค่าเบี่ยงเบนมาตรฐานคงที่ โดย คะแนนมาตรฐาน Z (Z-Score) คือคะแนนมาตรฐานที่มีลักษณะการกระจายเหมือนกับการกระจายของคะแนนดิบ มีค่าเฉลี่ยเป็นศูนย์และค่าเบี่ยงเบนมาตรฐานเป็นหนึ่ง สามารถคำนวณได้โดยใช้สูตรดังนี้

$$Z = \frac{X_i - \overline{X}}{S} \quad (2.6)$$

- $\overline{X}$ คือ ค่าเฉลี่ยของกลุ่มตัวอย่าง (Sample mean)
- Z แทนคะแนนมาฐานของแต่ละคน
- X แทนคะแนนดิบ
- S แทนส่วนเบี่ยงเบนมาตรฐาน (Standard Deviation)

2.6.5.1 คุณลักษณะของคะแนนมาตรฐาน Z

1. เป็นคะแนนที่มีค่าเฉลี่ยเป็นศูนย์และค่าเบี่ยงเบนมาตรฐานเป็นหนึ่ง

2. คะแนนมาตรฐานที่เป็นลบแสดงว่าคะแนนค่านั้นต่ำกว่าคะแนนเฉลี่ยและถ้าเป็นบวกแสดงว่าสูงกว่าค่าเฉลี่ย
3. การเปลี่ยนคะแนนดิบให้เป็นคะแนนมาตรฐาน ไม่จำกัดคะแนนเต็มของวิชาต่างๆ โดยปกติค่าคะแนนมาตรฐาน Z จะมีค่าอยู่ระหว่าง ±3
4. ลักษณะการกระจายเหมือนกับการกระจายของคะแนนดิบ

2.6.6 ทฤษฎีขีดจำกัดกลาง (The Central Limit Theorem)

ทฤษฎีขีดจำกัดกลาง (The Central Limit Theorem) ถือเป็นทฤษฎีที่เชื่อมโยงระหว่าง Normal distribution หรืออีกนัยหนึ่งคือ การกระจายของประชากร กับ Sampling distribution ซึ่งก็คือการกระจายที่ได้จาก ข้อมูลตัวอย่างและถือเป็นทฤษฎี ที่แก้ปัญหาความยุ่งยากซับซ้อน ของทฤษฎีต่างๆ ในสาขาสถิติเชิงอนุมาน (Inferential Statistics)

ทฤษฎีแนวโน้มเข้าสู่ศูนย์กลาง (Central Limit Theorem) อธิบายไว้ว่า " สำหรับประชากรใดๆแล้ว ถ้าเก็บตัวอย่างในจำนวนที่มากพอ การกระจายของค่าตัวอย่างดังกล่าวจะมีแนวโน้มใกล้เคียงกับการกระจายแบบธรรมชาติ (Normal distribution) เสมอ" ซึ่งเมื่อเราเก็บตัวอย่างมา แล้วหาค่าเฉลี่ย (X) และหากเรากะทำเช่นนี้หลายๆ ครั้ง แล้วนำเอา (X) ที่ได้หลายๆ ค่ามาพล้อตเป็นกราฟการกระจาย เราจะเห็นว่ากราฟที่ได้มีแนวโน้มเป็น Normal distribution เสมอ ไม่ว่าประชากรแม่ของข้อมูลตัวอย่างดังกล่าว จะมีการกระจายแบบใดก็ตาม โดยจุดสำคัญของทฤษฎีนี้ มีอยู่ 2 ข้อคือ

1. สามารถใช้ได้กับข้อมูลจากประชากรทุกประเภทการกระจาย ไม่จำกัดว่าจะมีการกระจายแบบธรรมชาติ (Normal distribution) เท่านั้น
2. จำนวนตัวอย่างที่เก็บมาจะต้องมากพอ นั่นแปลว่า จำนวนตัวอย่างมีผลต่อลักษณะการกระจายตัวของข้อมูล

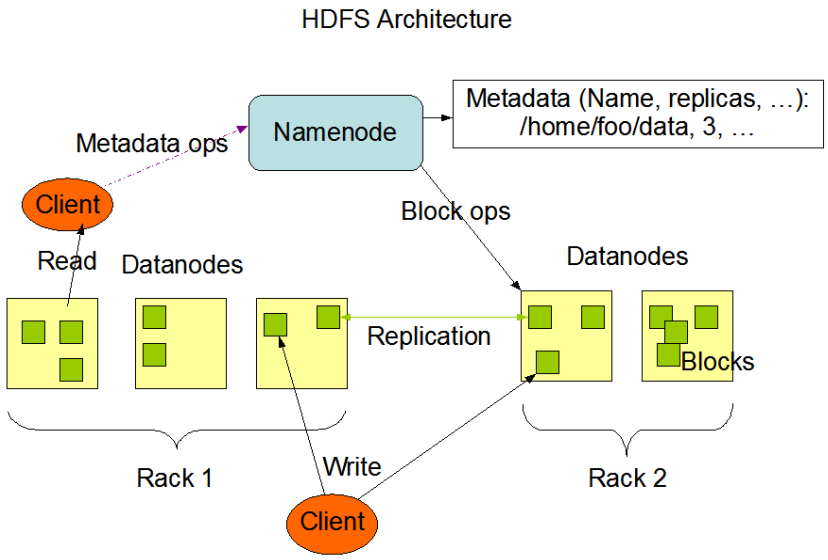
2.7 Hadoop

Hadoop Distributed File System [23] หรือ HDFS นั้นเป็น distributed file system ที่ออกแบบมาเพื่อทำงานบน commodity hardware ซึ่งคล้ายคลึงกับ distributed file system อื่นๆ แต่มีความแตกต่างจากระบบปฏิบัติการอื่นๆตรงที่ HDFS สามารถทนต่อการสูญเสียสูง (highly fault-tolerant) และออกแบบมาเพื่อรองรับการทำงานของ low-cost hardware โดยที่ HDFS จะมีความเหมาะสมสำหรับ application ที่มีข้อมูลขนาดใหญ่ ซึ่งสามารถมีได้ถึงหนึ่งหมื่นโหนด, หนึ่งร้อยล้านไฟล์ และมีขนาดใหญ่ได้มากถึงสิบ petabyte

โดยปกติแล้ว HDFS จะเป็นแบบ write-once-read-many access model โดยที่ไฟล์จะถูกเขียน เพียงครั้งเดียว โดยจะไม่สามารถแก้ไขไฟล์ได้ ทำให้เหมาะกับการเก็บข้อมูลประเภท Log ซึ่งจะทำให้ความสัมพันธ์ต่างๆของข้อมูลมีความง่ายมากขึ้น และสามารถเข้าถึงข้อมูลได้แบบ high throughput นอกจากนั้นยังมีการย้ายการประมวลผลให้ไปอยู่ใกล้กับที่เก็บข้อมูล ทำให้จะช่วยลดความคับคั่งของเครือข่าย (Traffic network) และยังช่วยเพิ่ม throughput ให้กับระบบได้อีกด้วย

2.7.1 สถาปัตยกรรมของ HDFS

HDFS นั้นจะมีลักษณะแบบ master/slave โดยจะมีโหนดที่สำคัญอยู่สองชนิด คือ NameNode หนึ่งตัว และ DataNode จำนวนหนึ่ง



ภาพที่ 2.6 สถาปัตยกรรมของ HDFS

NameNode: เป็น master server ที่คอยจัดการ file system namespace เช่น เปิด-ปิดไฟล์, เปลี่ยนชื่อไฟล์และสารบบ (directories) นอกจากนั้นยังทำหน้าที่ตัดสินใจในการ mapping block ต่างๆว่าจะเก็บเอาไว้ที่ DataNode ใด ซึ่งไฟล์และ directories จะถูกแสดงอยู่บน NameNode ในรูปของ inode ที่เก็บข้อมูลเกี่ยวกับ permissions, การแก้ไขต่างๆ และเวลาในการเข้าถึง, namespace และ disk space quotas โดยที่ไฟล์จะถูกแบ่งออกเป็น blocks และแต่ละ block จะถูกทำ replicate ไปตาม DataNode ต่างๆ (โดยปกติจะทำ replicate 3 โหนด แต่สามารถกำหนดเองได้ในแต่ละไฟล์) โดย NameNode จะจัดเก็บ namespace ทั้งหมดเอาไว้ใน RAM ซึ่งในแต่ละไฟล์จะมี inode data และรายชื่อของ blocks โดยจะจัดเก็บ metadata ทั้งหมดเอาไว้ใน Namespace image โดยจะถูกจัดเก็บเอาไว้ใน localhost ของ file system ที่เรียกว่า checkpoint นอกจากนั้น Namenode ยังมีการเก็บ Journal ซึ่งเป็น log ที่จะเก็บการเปลี่ยนแปลงต่างๆภายในระบบเอาไว้ใน local host file system และเพื่อเพิ่มความทนทาน จะมีการทำ redundant copies ของ checkpoint และ journal ที่ server อื่นๆเอาไว้ด้วย ซึ่งระหว่างที่ทำการ restart NameNode จะทำการ restore namespace โดยอ่านจาก namespace และ journal

DataNodes: เป็น one per node ภายใน cluster ซึ่งจะคอยจัดการการจัดเก็บข้อมูลไปยังโหนดที่ทำงานอยู่ นั่นคือทำหน้าที่ในการสร้าง-ลบ block และการทำ replication ซึ่งจะทำการรับ

คำสั่งจาก NameNode อีกที โดยในแต่ละ block replica บน DataNode จะประกอบด้วยไฟล์สองไฟล์ โดยไฟล์แรกจะเป็นข้อมูล และไฟล์ที่สอง คือ block ของ metadata ซึ่งรวมถึง checksum ของ block data และ block's generation stamp โดยขนาดของ data file จะเท่ากับความกว้างของ block จริงๆและไม่มีการร้องขอพื้นที่พิเศษเพื่อใช้ในการปัดให้ครบตามขนาดเหมือน file system แบบดั่งเดิม โดยในระหว่างการ startup แต่ละ DataNode จะเชื่อมต่อไปยัง NameNode และทำการ handshake เพื่อยืนยัน namespace ID และ software version ของ DataNode หากไม่ตรงกับ NameNode ก็จะทำการ shutdown อัตโนมัติเพื่อไม่ให้เกิด data corruption หรือ การสูญหายของข้อมูล หลังจากนั้นจะทำการส่ง block report ไปยัง NameNode ซึ่งจะประกอบด้วย block id, generation stamp และ ความยาวของแต่ละ block replica โดย block report จะทำการส่งทุกหนึ่งชั่วโมง โดยในระหว่างการทำงานปกติ DataNode จะส่ง heartbeats ไปยัง NameNode เพื่อยืนยันว่า DataNode นั้นยังทำงานอยู่และ block replicas ภายในก็ยังคงทำงานอยู่เช่นเดียวกัน ซึ่งข้อมูลที่ถูกส่งไปใน heartbeats นั้นจะมีข้อมูลเกี่ยวกับ ความจุ, storage ที่ถูกใช้งาน และจำนวนของ data transfer ที่กำลังทำงานอยู่ ซึ่งข้อมูลเหล่านี้จะใช้สำหรับการจองพื้นที่และการตัดสินใจในการทำ load balancing ของ NameNode โดยปกติจะทำ heartbeat ทุก 3 วินาที หาก NameNode ไม่ได้รับ heartbeat จาก DataNode ภายใน 10 นาที NameNode ก็จะทำการเปลี่ยนสถานะของ DataNode นั้นให้เป็น out-of-service และจะสร้าง replicas ที่อยู่ใน DataNode นั้นไปไว้ใน DataNode ใหม่

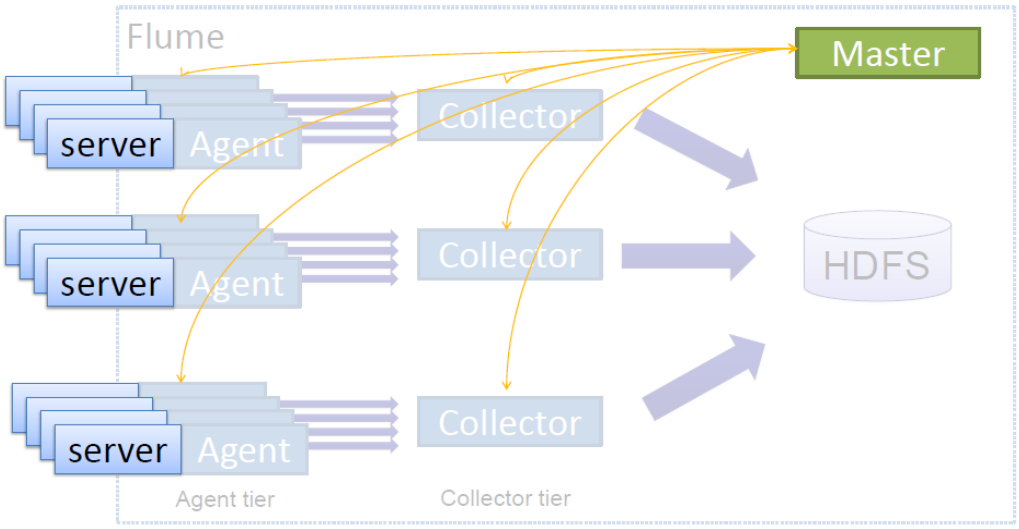
2.8 Flume

Cloudera Flume [24] เป็นโครงการที่มีจุดมุ่งหมายในการนำเสนอการผสมผสานการทำงานของ Hadoop และข้อมูลในลักษณะ Streaming Data สำหรับการเก็บรวมรวมข้อมูลล็อก โดยเป็น Framework ลักษณะของรางน้ำในการเก็บรวบรวมข้อมูลล็อก ของทุกๆ machine ที่อยู่ในระบบ จากนั้นจึงทำการส่งต่อข้อมูลอย่างรวดเร็วจากหลายๆแหล่งไปยังศูนย์กลางที่เก็บข้อมูล เช่น Hadoop Distributed File System (HDFS) เพื่อนำมาวิเคราะห์หรือตรวจสอบความผิดปกติและพฤติกรรมในการทำงานที่เกิดขึ้น

Flume จะมีทั้งความเป็น Distributed, Reliable และ Available service ของการรวบรวม log ได้อย่างมีประสิทธิภาพและสามารถเคลื่อนย้าย log ที่มีขนาดใหญ่ โดยเป็นทั้ง Simple และ Flexible architecture ที่เป็นแบบ Streaming dataflow ที่สามารถบริหารจัดการกับระบบได้แบบศูนย์กลาง (Centralize)

Flume’s Architecture

สถาปัตยกรรมของ Flume จะมีลักษณะ Stream-oriented data flow ซึ่ง data flow จะมีเส้นทางที่ต้องส่งไปต่อยัง destination และประกอบด้วย Logical node ที่สามารถส่งต่อหรือรวบรวม event ที่ได้รับมา ซึ่งสามารถ Configure ได้จากส่วนของ Flume master ที่เป็นตัวควบคุมทั้งหมด



ภาพที่ 2.7 แสดงถึงสถาปัตยกรรมของ Flume

ภาพที่ 2.7 แสดงถึงการใช้งานทั่วไปในการเก็บรวบรวมข้อมูล log จาก Server ซึ่งสามารถแบ่งออกเป็นสามส่วน คือ Agent, Collector และ Storage โดย Agent tier เป็นจุดแรกที่จะมีการเชื่อมต่อกับ Flume ที่ถูก install ให้เป็นส่วนของ node ในแต่ละ machine ที่จะมีการ generate log เพื่อส่งต่อไปยังส่วนต่อไป คือ Collector tier ซึ่งเป็นส่วนรวมข้อมูลของแต่ละ data flow จากนั้นจึงมีการส่งต่อไปยัง storage tier เป็นส่วนสุดท้าย ยกตัวอย่างเช่น ส่วน Agent อาจจะมีการเก็บ syslog หรือ monitor log ของ Service เช่น web server หรือ Hadoop JobTracker แล้วทำการส่งต่อไปยัง collector จากนั้น collector ก็จะรวบรวมข้อมูลจากทุกๆ agent และทำการ stream ข้อมูลต่อไปยัง storage tier เช่น HDFS ได้

2.9 SEC (Simple Event Correlator)

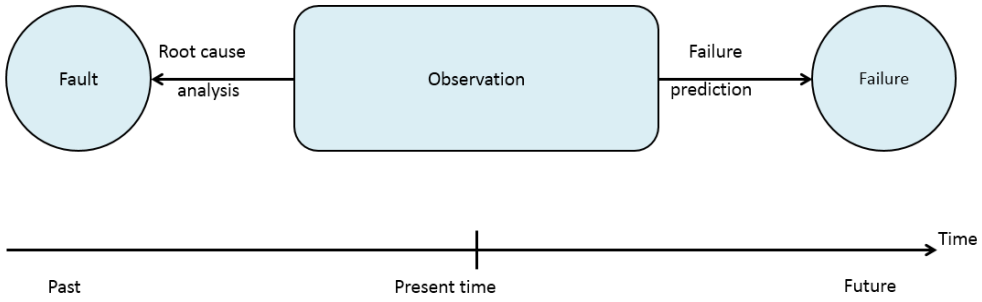
Simple Event Correlator (SEC) [25] เป็นโอเพนซอร์สที่ใช้สำหรับการทำ Event Correlation เพื่อวิเคราะห์ข้อมูล Log โดยทำงานแบบ Stream event สำหรับการตรวจจับในลักษณะ Rule-based approach ซึ่งใช้วิธี Regular expression เพื่อทำการ Match หรือตรวจจับรูปแบบเพื่อระบุความผิดปกติของข้อมูล Log จากนั้นจึงทำการกำหนด Action ที่ต้องการ ซึ่ง Action สามารถสร้างข้อความ Log, สร้างไฟล์ Event ใหม่, ทำการ Execute กับโปรแกรมอื่น และยังสามารถสร้างความสัมพันธ์กับส่วนต่างๆในลักษณะ Event ที่ซับซ้อนได้

2.10 งานวิจัยที่เกี่ยวข้อง

วิธีการตรวจสอบความน่าเชื่อถือของระบบด้วยการวิเคราะห์ Failure จากข้อมูลล็อกนั้น ได้มีงานวิจัยจำนวนมากที่ได้กล่าวถึงการพัฒนาและเพิ่มประสิทธิภาพในการวิเคราะห์ Failure

2.10.1 Failure analysis

จากงานวิจัย [26] ได้กล่าวถึงคุณลักษณะที่แตกต่างกันระหว่าง Failure prediction และ Root cause analysis โดยคุณลักษณะของทั้งสอง มีการตรวจสอบข้อมูลจากการวิเคราะห์ Log เหมือนกัน แต่มีความแตกต่างกันดังภาพที่ 2.8

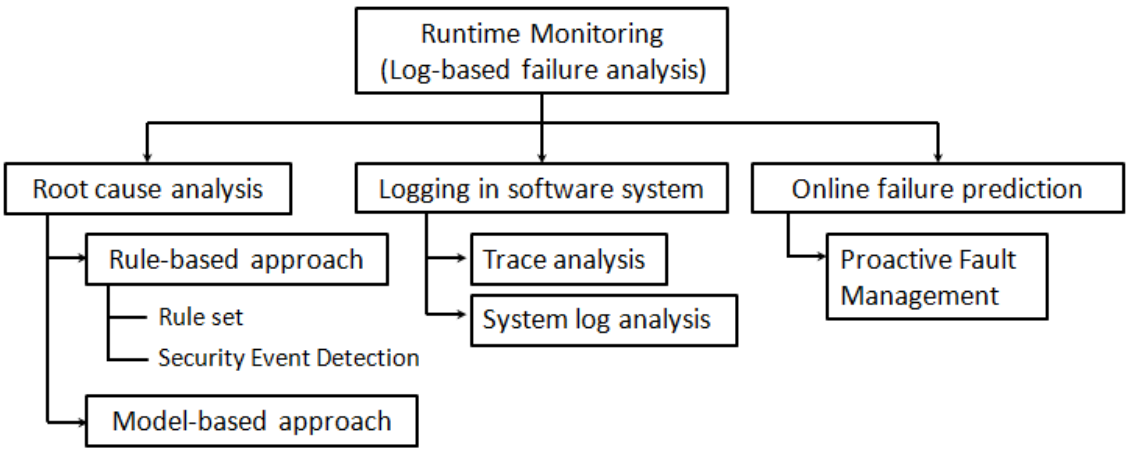


ภาพที่ 2.8 แสดงความแตกต่างระหว่าง Root cause analysis และ Online failure prediction [26]

จากภาพที่ 2.8 Root cause analysis จะพยายามระบุสาเหตุของปัญหาและหาเหตุผลว่าส่วนใดที่เป็นต้นเหตุของการเกิด Failure ในระบบ ในขณะที่ Failure prediction จะมีลักษณะเป็นการคาดเดาเหตุการณ์ที่อาจจะเกิดขึ้นในอนาคตและประเมินความเสี่ยงของพฤติกรรมที่เปลี่ยนไปจนทำให้เกิด Failure เช่น ประเมินว่าการให้บริการของระบบนั้นมีเครื่องสำรองที่ทำให้ระบบยังสามารถทำงานได้อยู่หรือไม่ หรือ Load การทำงานของระบบเป็นอย่างไรในช่วงเวลานั้นๆ เป็นต้น

ยกตัวอย่างเช่น กรณีที่ตรวจสอบพบว่า Database นั้นมีสถานะเป็น Unavailable หากเป็นกรณีของ Root cause analysis จะพยายามหาสาเหตุว่าเพราะเหตุใดทำให้ Database นั้นใช้งานไม่ได้ เช่น Network ของระบบมีปัญหา, Configure ผิดพลาด แต่หากเป็นกรณีของ Failure prediction จะพยายามประเมินสถานการณ์ว่าระบบจะสามารถให้บริการต่อไปได้หรือไม่ โดยจะตรวจสอบว่าระบบมี Backup database, มีกลไก fault tolerance, หรือ load ในระบบตอนนี้เป็นอย่างไรบ้าง เป็นต้น

การวิเคราะห์ Log ในระบบที่มีความซับซ้อน การใช้ Failure Prediction อาจยากต่อการสร้าง Algorithm เพื่อคืนผลลัพธ์ของการคาดคะเนกลับมา และอาจได้ผลลัพธ์เป็นจำนวนมาก ทำให้ผู้ดูแลต้องรับหน้าที่ในการกรองข้อมูลเพื่อหาผลลัพธ์ที่อาจเกิดขึ้นที่แท้จริง ซึ่งการที่จะพยากรณ์เพื่อซ่อมแซมและแก้ไข Service ใด Service หนึ่ง อาจไม่ได้ส่งผลให้ภาพรวมของ Service สามารถทำงานได้ถูกต้องก็เป็นได้ เนื่องจาก Service นั้นไม่ได้เป็นต้นตอของปัญหาที่แท้จริง



ภาพที่ 2.9 แสดงวิธีตรวจสอบการวิเคราะห์ Failure จากข้อมูล Log แบบต่างๆ

จากภาพที่ 2.9 จะแสดงให้เห็นถึงความแตกต่างระหว่าง Root cause analysis และ Online failure prediction ดังนั้น จึงสามารถแบ่งออกเป็น 3 รูปแบบใหญ่ๆ ได้แก่

2.10.2 Online failure prediction

ในงานวิจัย [14, 15] นำเสนอเรื่อง Proactive self adaptation โดยเป็นกลไกในการทดสอบระบบแบบ Online เพื่อตรวจจับพฤติกรรมที่เปลี่ยนแปลงไปของระบบก่อนที่ระบบจะเกิดความล้มเหลว โดยจะเป็นลักษณะของการประเมินความเสี่ยงที่อาจเกิดขึ้นกับระบบ แต่ไม่ได้หาต้นตอของปัญหาที่เกิดขึ้นกับระบบที่แท้จริง

2.10.3 Logging in software system

จะเป็นลักษณะการตรวจสอบในระบบโดยตรงจาก log ที่เกิดขึ้น เช่น ในงานวิจัย [8] กล่าวถึงการใช้คำสั่ง grep ซึ่งเป็นคำสั่งที่ช่วยในการตรวจสอบและ Filter ข้อมูลเพื่อวิเคราะห์ข้อมูล log ที่ผิดปกติ ซึ่งใช้งานง่าย สะดวก และรวดเร็ว แต่จะเห็นว่าข้อเสียของการวิเคราะห์ในลักษณะนี้คือเมื่อเกิดปัญหากับระบบการให้บริการ ผู้ดูแลจะต้องรับมือและจัดการกับปัญหาเองทุกครั้ง, [9] กล่าวถึงวิธีในการ Filter เพื่อลดจำนวนข้อมูล Log ที่ไม่จำเป็นหรือไม่เกี่ยวข้องในการวิเคราะห์ Failure เพื่อนำเฉพาะส่วนที่จำเป็นมาใช้และแก้ไขปัญหาทางด้านความจุที่มีจำกัด แต่ไม่ได้มีการ Correlate เพื่อวิเคราะห์สิ่งที่เกิดขึ้นจริงๆ และการลดจำนวน log ที่ไม่จำเป็นนั้น อาจส่งผลให้เกิดความผิดพลาดในการวิเคราะห์ Failure ก็เป็นได้

2.10.4 Root Cause Analysis

Rule-based approach จะเป็นการกำหนด Rule เพื่อตรวจจับสิ่งผิดปกติที่เกิดขึ้น ในงานวิจัย [5, 11] ใช้วิธี rule-based และ case-based ในการเพิ่มประสิทธิภาพและกระบวนการอัตโนมัติโดยมีการ correlate event เพื่อลดและหาความสัมพันธ์ของข้อมูลเพื่อให้ได้ความหมายที่แท้จริง หาก Rule-base ไม่สามารถจัดการกับ event ที่เข้ามาได้ จะมีการส่งต่อไปยัง Case-based

เพื่อช่วยในการจัดการและเพิ่มประสิทธิภาพมากยิ่งขึ้น แต่อย่างไรก็ตาม จะเห็นว่าในงานวิจัยนี้ไม่ได้คำนึงถึงข้อมูลที่มีขนาดใหญ่ในเชิง Bid data, [12, 13] ใช้วิธีพัฒนาการผลิต log เพื่อที่จะเพิ่มประสิทธิภาพในการวิเคราะห์ Failure โดยเพิ่ม Logging rule เพื่อตรวจสอบ Availability และ Reliability ของระบบ แต่ในการพัฒนาการผลิต log นั้น จำเป็นที่จะต้องแก้ไข code เพื่อทำให้เป็นไปตาม logging rule ที่ต้องการ ซึ่งอาจส่งผลกระทบอื่นๆตามมาก็เป็นได้

ในงานวิจัย [27] ได้ใช้วิธี Rule-based approach ในการวิเคราะห์ข้อมูล Log ในด้านความปลอดภัยของระบบ รวมถึงการออกแบบการใช้ Event Correlation เพื่อวิเคราะห์ในลักษณะ Distributed เพื่อเพิ่มความยืดหยุ่นในการลดขนาดของข้อมูล Log ที่อาจทำให้เกิดปัญหาทางด้าน Network traffic.

Model-based approach [28, 29] อุปกรณ์แต่ละตัวในระบบจะถูก model ให้มี attribute, behavior และมีความสัมพันธ์กับ model ในส่วนอื่นๆ ซึ่งพฤติกรรมทั้งหมดในระบบจะเป็นผลลัพท์จากการกระทำกันระหว่าง Component model โดย Event correlation ที่ได้ คือผลของการหาการจำลอง model ที่สัมพันธ์กัน แต่สำหรับลักษณะงานแบบ Service-oriented จะมองในมุมที่ขึ้นกับพฤติกรรมของผู้ใช้บริการด้วย จึงอาจมีพฤติกรรมของ Service ในบางสถานการณ์ที่ยากต่อการอธิบาย รวมถึง Rule-based จะมีความยืดหยุ่นและง่ายต่อการกำหนด rule มากกว่า เพราะปัญหาที่เกิดนั้นสามารถเพิ่ม เปลี่ยนหรือลบ ได้ง่ายและบ่อยครั้งตามที่ต้องการจากผู้ใช้งานหรือผู้ดูแลระบบก็เป็นได้

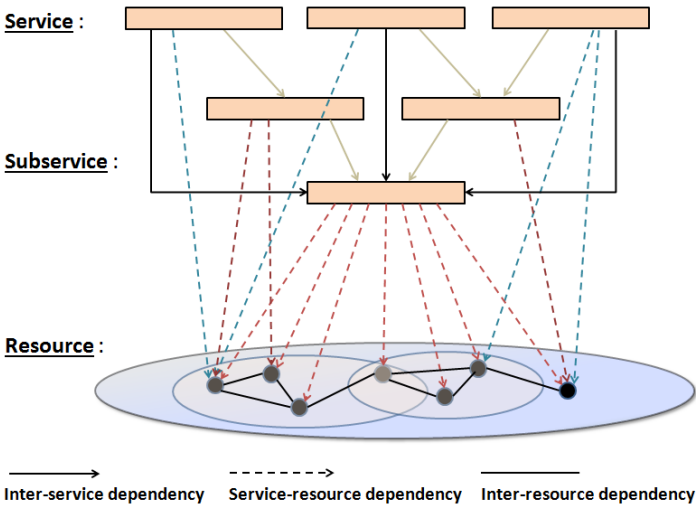
บทที่ 3
โครงสร้างระบบการตรวจสอบแบบการหาความสัมพันธ์ข้อมูล

3.1 โครงสร้างทั่วไปของระบบตรวจสอบ

ในส่วนงานวิจัยนี้ได้รวบรวมการออกแบบการเก็บข้อมูล (Data collection) สำหรับการ Monitoring และการวิเคราะห์ (Analyzing) ในระบบที่มีลักษณะเป็น Large distributed system ซึ่งต้องรับมือกับปริมาณข้อมูลจำนวนมากในแบบเรียลไทม์ โดยมีเครื่องมือสำหรับ Monitoring จำนวนมากที่ออกแบบมารองรับสภาพแวดล้อมการทำงานแบบ Large scale [24], [30], [31], [32] โดยวิเคราะห์จากข้อมูลล็อก ซึ่งจากภาพที่ 2.9 จะเห็นว่าจัดอยู่ในส่วนของ Root cause analysis เนื่องจากในงานวิจัยนี้ไม่ได้มีการประเมินความเสี่ยงและสถานการณ์ที่อาจเกิดขึ้น แต่จะใช้กระบวนการวิเคราะห์ถึงต้นตอของสาเหตุปัญหาในลักษณะ Rule-based approach เพื่อทำการ Match ข้อมูล Log จากนั้นจึงทำการกำหนด Action ที่ต้องการ ทำให้เกิดความยืดหยุ่นและง่ายต่อการกำหนด rule ซึ่งในส่วนของปัญหาเรื่องของปริมาณ Log ที่เพิ่มมากขึ้นเรื่อยๆดังที่ได้กล่าวไปแล้วในงานวิจัยอื่นๆนั้น ในงานวิจัยนี้จะนำสถาปัตยกรรม HDFS เข้ามาช่วยในการเก็บข้อมูลประเภท log เพื่อรองรับปริมาณ log ที่เกิดขึ้นระหว่างการทำงาน โดยมองลักษณะการทำงานของระบบเป็น Service-oriented ซึ่งความหมายของ Fault Management ในระดับ Resource และ Service จะมีลักษณะที่ต่างกันดังนี้ [11]

Resource fault management: จะมองในลักษณะของ Resource-oriented และจัดการกับ Event, fault และ Error ที่เกิดขึ้นใน Network และ End system ซึ่งอาจจะไม่ได้มีผลกระทบต่อประสิทธิภาพในการทำงานของ Service โดยตรง ดังนั้น จะเห็นว่าอาจจะไม่สามารถการันตีได้ถึงคุณภาพของระดับการให้บริการ SLAs

Service fault management: จะมีลักษณะในทางตรงกันข้ามกับ Resource fault management คือ Service fault management จะมองถึงคุณภาพในการให้บริการโดยคำนึงถึงมุมมองของผู้ใช้บริการ ดังนั้นการเสื่อมคุณภาพในการให้บริการจะต้องสามารถระบุไปถึง Resource หรือ Subservice ที่เป็นปัญหาได้



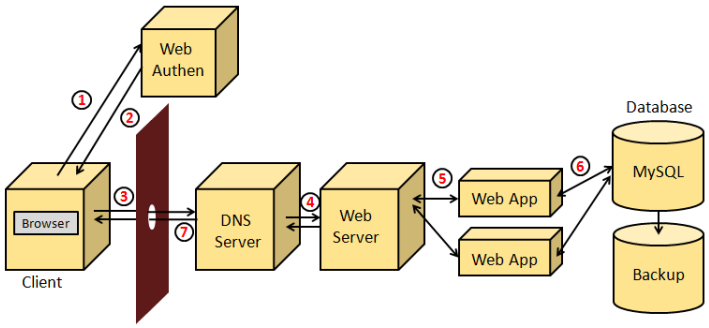
ภาพที่ 3.1 แสดงลักษณะของการขึ้นต่อกันระหว่าง Service, Subservice และ Resource [33]

Service: ในที่นี้หมายถึงระบบการทำงานที่ให้บริการจากผู้ให้บริการไปยังผู้ใช้บริการ ซึ่งคุณภาพของการให้บริการจะเป็นไปตาม SLAs ที่ควรจะเป็น และการทำงานของ Service จะขึ้นอยู่กับ Service อื่นๆที่เราเรียกว่า Subservice และ Resource ทำงานร่วมกัน

Subservice: เป็น Service อื่นๆที่ใช้งานเพื่อให้ Service หลักสามารถใช้งานได้ โดยในที่นี้เราเรียกว่า Subservice โดยหาก Subservice บางอย่างไม่สามารถใช้งานได้ อาจไม่มีผลกระทบต่อการให้บริการของ Service โดยตรง หรืออาจจะมีผลกระทบต่อการให้บริการของ Service หลัก ซึ่งอาจเป็นผลให้ Service สามารถให้บริการได้ไม่เต็มประสิทธิภาพหรืออาจจะไม่สามารถให้บริการได้เลยก็เป็นได้ จากในรูปภาพที่ 1 จะเห็นว่า Subservice ไม่จำเป็นที่จะต้องใช้งานสำหรับ Service เพียง Service เดียว อาจจะให้บริการ Service อื่นๆด้วยก็เป็นได้

Resource: เป็นอุปกรณ์ที่ถูกใช้เพื่อให้ Service สามารถทำงานได้ ซึ่งการ Failure ของ Service อาจจะเกิดจาก Resource ก็เป็นได้ เช่น Network link, End system, Main memory, Hard disk drive, Application process เป็นต้น

ในงานวิจัยนี้ได้ทำการทดลองกับระบบ Web Application ของคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์ ซึ่งมีลักษณะโครงสร้างการทำงานหลักดังภาพที่ 3.2

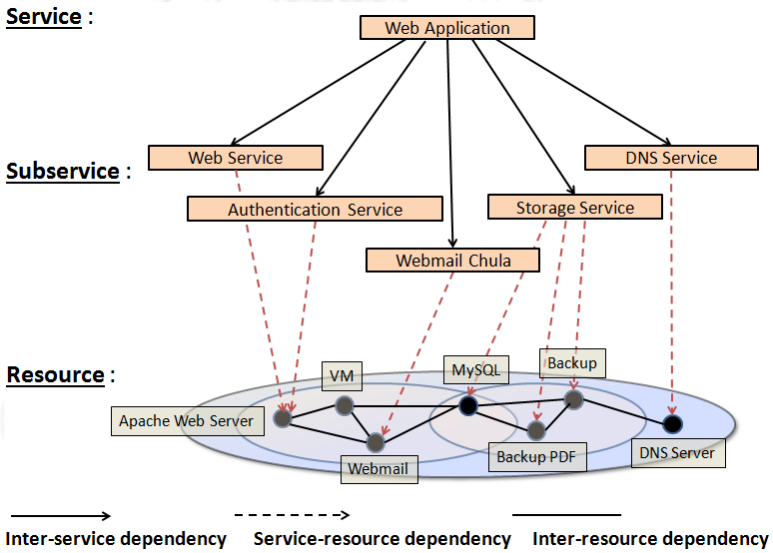


ภาพที่ 3.2 แสดงโครงสร้างของ Web Application

จากภาพที่ 3.2 แสดงถึงโครงสร้าง Diagram ของ Web application ที่ใช้งาน โดยลำดับการทำงานจะเป็นดังต่อไปนี้

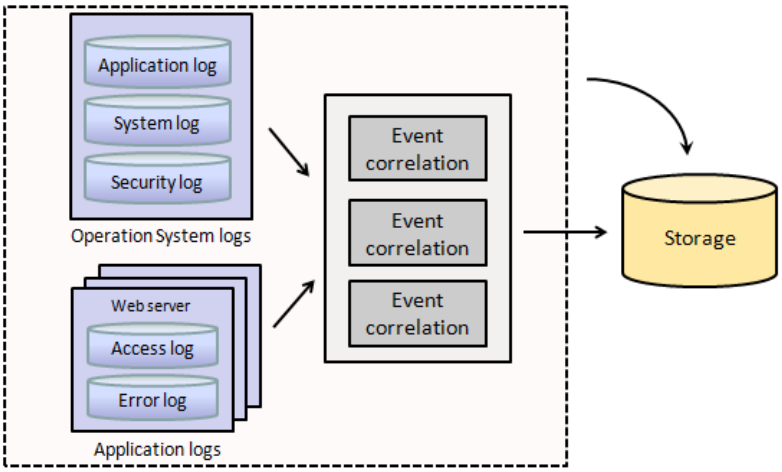
1. Client ทำการร้องขอผ่าน Browser เพื่อเข้าใช้งาน โดยต้องทำการใส่ Username และ Password เพื่อยืนยันตัวตน
2. ตรวจสอบ Authentication และ Log on user เพื่อให้สิทธิในการใช้งาน
3. ร้องขอไปยัง DNS Server
4. เชื่อมต่อร้องขอข้อมูลต่อไปยัง Web server
5. Web server มีการประมวลผลและส่งข้อมูลกลับไปยัง Browser หากเป็นข้อมูลรูปภาพ, CSS หรือ Static web page ที่สามารถอ่านจาก Disk และส่งข้อมูลกลับไปยัง Browser ได้ทันที แต่หากเป็น Dynamic page อาจมีการส่งต่อไปยัง Application อื่นๆเพื่อประมวลผล
6. Web application อาจมีการเรียกใช้หรือร้องขอข้อมูลจาก Database server
7. มีการส่งข้อมูลกลับไปยัง Browser เพื่อแสดงผล

ดังนั้น จะเห็นว่าการให้บริการ Web Application ได้นั้น จำเป็นจะต้องเกิดจาก Service ต่างๆทำงานร่วมกัน และเมื่อทำการพิจารณาและแบ่งระดับของ Service, Subservice และ Resource จะได้ภาพที่ 3.3



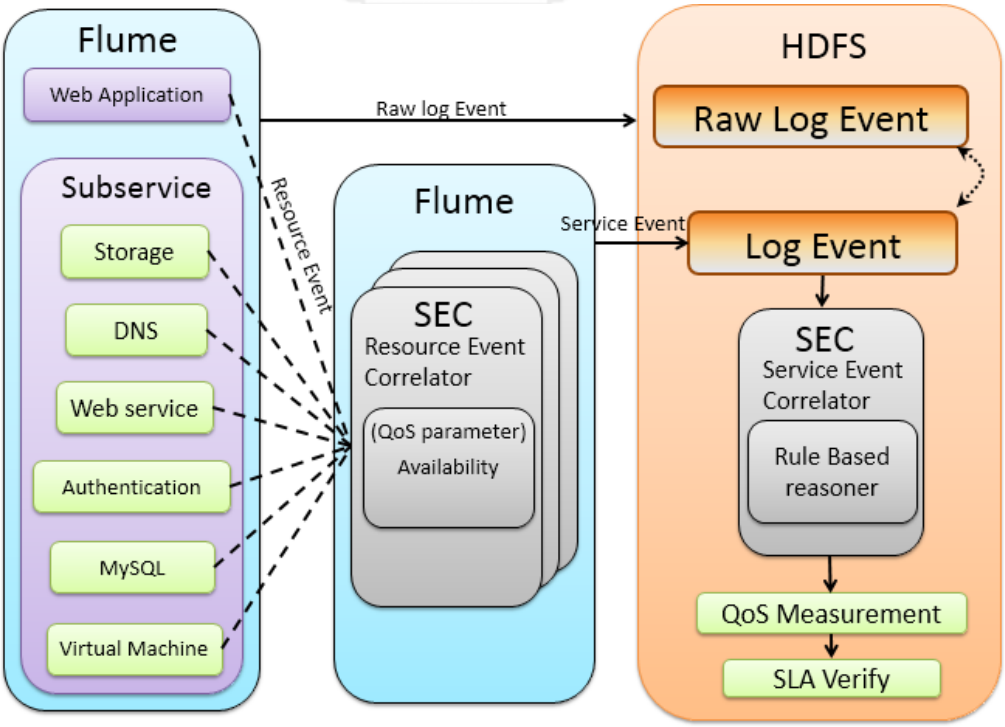
ภาพที่ 3.3 แสดงการแบ่งระดับของ Web Application Service

จากภาพที่ 3.3 จะเห็นว่า Service ที่ให้บริการนั้นเป็น Web application ซึ่ง Subservice ของ Web application ในที่นี้คือ Storage Service ที่มีไว้เพื่อเก็บข้อมูลต่างๆระหว่างการทำงาน, Authentication Service สำหรับระบุตัวตนในการใช้งาน, DNS Service ใช้เพื่อหาที่อยู่ของ Web page อีกทั้งยังมีส่วนของ Virtual Machine ที่ใช้ร่วมกันในการทำงาน เป็นต้น



ภาพที่ 3.4 แสดงการทำงานแบบ Distributed Event correlation

จากภาพที่ 3.4 แสดงถึงแนวคิดของการทำงานแบบ Distributed Event correlation โดยในระบบที่มีขนาดใหญ่และซับซ้อนนั้น การวิเคราะห์ข้อมูล log ผ่าน Event correlation เพียงตัวเดียวอาจจะไม่สามารถรองรับการ Scale ของระบบได้ดีเท่าที่ควร เนื่องจากมีข้อจำกัดทางด้านความแม่นยำและความรวดเร็วในการ Detect ผ่าน Event correlation ที่มีข้อมูล log จำนวนมากไหลผ่าน ซึ่งอาจทำให้เกิดการตกหล่นในการ Detect ข้อมูล ดังนั้นในงานวิจัยนี้จะนำ Distributed Event correlation มาใช้ในการเก็บรวบรวมข้อมูล Event log เพื่อวิเคราะห์ Availability ของระบบ ซึ่งจะมีลักษณะดังภาพที่ 3.5



ภาพที่ 3.5 Framework ของการทำงานภายในระบบ

จากภาพที่ 3.5 จะเห็นว่ามีลักษณะ Event เป็น 3 ส่วนคือ

1. Raw log event.
2. Resource Event.
3. Service Event.

Raw log event: คือ Event log ที่เกิดจากการ Generate จาก Resource ในทุกๆส่วนที่สนใจ โดยไม่มีการผ่านการ Correlation แต่อย่างใด ซึ่งการเก็บ Event log ในลักษณะนี้จะใช้เครื่องมือ Flume เป็นตัวโอนถ่ายข้อมูล Log เพื่อไปเก็บยัง HDFS เพื่อให้สามารถรองรับการทำงานของ Big data และตรวจสอบหรือวิเคราะห์ในภายหลังได้ เนื่องจากหากเราใช้ Hard drive เป็นตัวเก็บข้อมูล จะต้องใช้เวลาอย่างมากในการอ่าน เขียนและโอนถ่ายข้อมูล ในขณะที่ HDFS เป็นการทำงานแบบ Parallel จึงทำให้เวลาที่ใช้มีความรวดเร็วมากกว่า รวมถึงข้อดีต่างๆ เช่น Reliability, Scalability และ Highly fault tolerant


```
{"body":"bprd\t\t13720/tcp\t\t\t# VERITAS NetBackup","timestamp":1333781200105,"pri":"INFO","nanos":1654138220415917,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u00002ÃœPÂ¬","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"bprd\t\t13720/udp","timestamp":1333781200105,"pri":"INFO","nanos":1654138220489260,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000Ã-\u0000\u0000Â^","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"bpdbm\t\t13721/tcp\t\t\t# VERITAS NetBackup","timestamp":1333781200105,"pri":"INFO","nanos":1654138220570899,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000ÃžÃ¤Ã¬","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"bpdbm\t\t13721/udp","timestamp":1333781200105,"pri":"INFO","nanos":1654138220642774,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000Ã¯Â¼Ã^Âœ","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"bpjava-msvc\t13722/tcp\t\t\t# BP Java MSVC Protocol","timestamp":1333781200105,"pri":"INFO","nanos":1654138220734012,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000ÃƒÂ Â Ã ","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"bpjava-msvc\t13722/udp","timestamp":1333781200105,"pri":"INFO","nanos":1654138220816410,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000ÃƒdÂ-Ã;","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
{"body":"vnetd\t\t13724/tcp\t\t\t# Veritas Network Utility","timestamp":1333781200105,"pri":"INFO","nanos":1654138220899477,"host":"Andmebaas","fields":{"AckTag":"20120407-134639031+0700.1654137147086957.00000031","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000Ã²KÎÂ«","rolltag":"20120424-163240405+0700.350874908287456.00000073"}}
```


ภาพที่ 3.6 แสดงตัวอย่าง Raw log event ที่มาจาก Flume

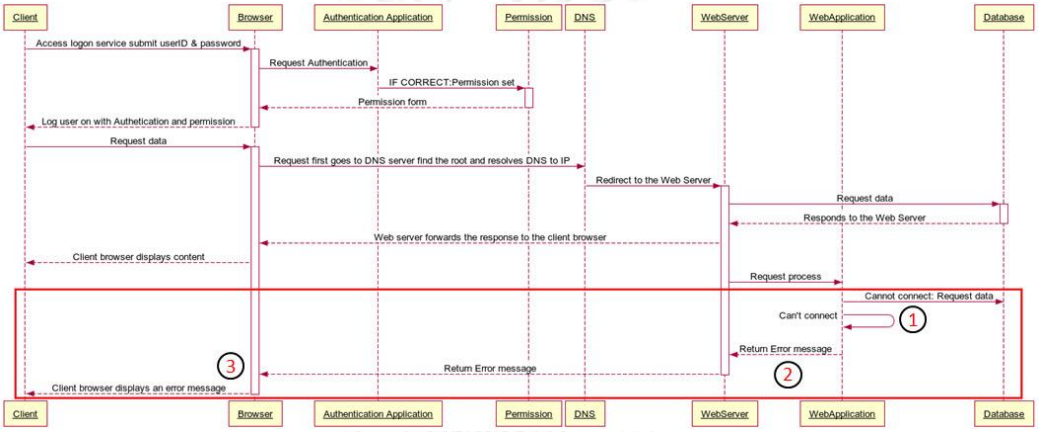
จากภาพที่ 3.6 แสดงถึงตัวอย่างของ raw log event โดยในที่นี้เป็นตัวอย่างของ syslog ที่เกิดขึ้นในระบบ ซึ่งจะมีในส่วนของ *body* เป็นข้อความที่แสดงถึงสิ่งที่เกิดขึ้นและเป็นข้อมูลของ event, *timestamp* เป็นเวลาที่ log ได้ถูกสร้างและบันทึก และอาจมีส่วนอื่นๆ ที่รวมอยู่ในข้อมูลด้วย เช่น PID ของ process, IP address ของอุปกรณ์ที่มีการสร้างข้อมูล log รวมถึงระดับความรุนแรงที่เกิดขึ้นใน event

Resource Event: คือ Event log ที่เกิดจากการ Generate จาก Resource ในแต่ละส่วน เพื่อที่จะนำข้อมูล log มาทำการ Correlation ผ่านเครื่องมือตัวอย่าง คือ Simple event correlator (SEC) และทำการวัดผลออกมาให้อยู่ในรูปแบบของ Availability เพื่อนำไปสู่ Service Event อีกทีหนึ่ง ซึ่งเพื่อให้ระบบสามารถ scale ได้ เราสามารถกำหนดให้ SEC ทำงานได้มากกว่า 1 ตัว

Service event: คือ Event log ที่จะสื่อถึงประสิทธิภาพของการให้บริการ ซึ่ง Service Event จะสื่อถึงมุมมองของผู้ใช้บริการว่าการให้บริการยังคงอยู่ในสภาพพร้อมใช้งานและมีการเสื่อมคุณภาพหรือไม่

QoS Parameters: สำหรับ Service นี้จะมี QoS parameters เป็น Availability และ Detection time ในการตรวจสอบเพื่อให้เป็นไปตาม SLA

โดยการตรวจสอบ Availability สามารถตรวจสอบได้จากข้อมูลล็อกของแต่ละ Resource, Subservice และ Service ซึ่งหากการทำงานในแต่ละส่วนส่งผลกระทบต่อการให้บริการของระบบ โดยที่เป็นการทำงานในลักษณะขึ้นต่อกันระหว่างการให้บริการและอุปกรณ์ เนื่องจากอุปกรณ์เพียง 1 อุปกรณ์ (ไม่ว่าจะเป็น Software หรือ Hardware) อาจส่งผลให้อุปกรณ์อื่นๆภายในระบบเกิดปัญหาและไม่สามารถให้บริการต่อได้



ภาพที่ 3.7 Sequence Diagram แสดงตัวอย่างการ Failure ของระบบ

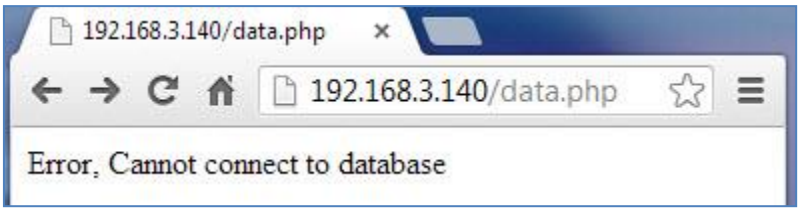
จากภาพที่ 3.7 แสดงถึงตัวอย่างการ Failure ของระบบ โดยการทำงานปกตินั้น เมื่อ Client ทำการร้องขอการเข้าใช้งานระบบ จะมีขั้นตอนการทำงานดังรูปจะเห็นว่าเมื่อระบบไม่สามารถเชื่อมต่อไปยัง Database ซึ่งเป็น Subservice ของ Web application ได้ ทำให้การให้บริการมีสถานะเป็น unavailability คือไม่สามารถให้บริการหรือแสดงผลลัพธ์ที่ถูกต้องไปยังผู้ใช้บริการได้นั้นเอง โดยรูปแบบการ Correlation อาจเป็นดังนี้

1. มีการสร้าง Event MySQL_DOWN เกิดขึ้น เมื่อพบว่าไม่สามารถเชื่อมต่อไปยัง Database ได้
2. อาจเกิด Propagation ขึ้นคือมีการสร้าง Event WEBAPP_PROBLEM และ WEBSERVER_PROBLEM (เนื่องจาก MySQL_DOWN)

```
[Mon Jan 07 00:10:33 2013] [error] [client 192.168.3.1] PHP Warning:  mysql_connect(): Can't connect to local MySQL server th
rough socket '/var/run/mysqld/mysqld.sock' (2) in /var/www/data.php on line 8
[Mon Jan 07 00:10:33 2013] [error] [client 192.168.3.1] File does not exist: /var/www/favicon.ico
[Mon Jan 07 00:11:39 2013] [error] [client 192.168.3.133] PHP Warning:  mysql_connect(): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (2) in /var/www/data.php on line 8
[Mon Jan 07 00:11:39 2013] [error] [client 192.168.3.133] File does not exist: /var/www/favicon.ico
[Mon Jan 07 00:11:39 2013] [error] [client 192.168.3.133] File does not exist: /var/www/favicon.ico
[Mon Jan 07 00:15:37 2013] [error] [client 127.0.0.1] PHP Warning:  mysql_connect(): Can't connect to local MySQL server thro
ugh socket '/var/run/mysqld/mysqld.sock' (2) in /var/www/data.php on line 8
```

ภาพที่ 3.8 แสดงปัญหา Web server ไม่สามารถเชื่อมต่อไปยัง Database

1. ผลที่ตามมาคือทำให้ Client ไม่สามารถเข้าใช้งานได้



ภาพที่ 3.9 Client ไม่สามารถเข้าใช้งานได้

2. เมื่อ Event DATABASE:DOWN ซึ่งเป็นต้นตอของปัญหาถูกแก้ไข จึงทำให้ Event อื่นๆสามารถกลับมาใช้งานได้

```
Jan  7 00:21:10 ubuntu2 kernel: [ 4956.765326] type=1400 audit(1357546870.644:23): apparmor="STATUS" operation="profile_repla
ce" name="/usr/sbin/mysqld" pid=3087 comm="apparmor_parser"
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3110]: Upgrading MySQL tables if necessary.
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3113]: /usr/bin/mysql_upgrade: the '--basedir' option is always ignored
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3113]: Looking for 'mysql' as: /usr/bin/mysql
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3113]: Looking for 'mysqlcheck' as: /usr/bin/mysqlcheck
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3113]: This installation of MySQL is already upgraded to 5.1.63, use --force
if you still need to run mysql_upgrade
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3120]: Checking for insecure root accounts.
Jan  7 00:21:11 ubuntu2 /etc/mysql/debian-start[3124]: Triggering myisam-recover for all MyISAM tables
```

ภาพที่ 3.10 แสดงสถานะ Database เมื่อเชื่อมต่อการทำงานตามปกติ

จากตัวอย่างการ Failure ของระบบที่มีสาเหตุมาจากการเชื่อมต่อไปยัง MySQL ไม่ได้นั้น สามารถกำหนด rule ผ่าน SEC เพื่อทำการ Detect ปัญหาและกำหนด Action ต่างๆดังรูป

```
# ------------------------------------------------------------------------
# Sample matching input lines:
# [Mon Jan 07 00:10:33 2013] [error] [cliect 192.168.3.1] PHP Warning:
# mysql_connect(): Can't connect to local MySQL server through socket
# '/var/run/mysql/mysqld.sock' (2) in /var/www/data.php on line 9
# ------------------------------------------------------------------------

type=single
ptype=RegExp
pattern=\[error\] \[client [\d\.]+\] PHP Warning: \
 mysql_connect\(\): Can't connect to local MySQL server through socket
desc=Can't connect to local MySQL server
context=!TIMER
action=write /usr/local/sec/sec_out/ERR0001-database.out MySQL is Down at %t; \
      event MySQL_DOWN ; \
      spawn /usr/local/sec/conf/ERR0001-database-CheckStatus.sh ; \
      create TIMER 10;

# This rule will match part of the output of ERR0001-database-CheckStatus.sh
type=Single
ptype=RegExp
pattern=MySQL is Up
desc=MySQL is Up
action=write /usr/local/sec/sec_out/ERR0001-database.out '%s' at %t

# If event MySQL_DOWN more than 5 in 60 minutes
type=SingleWithThreshold
ptype=RegExp
pattern=MySQL_DOWN
desc=MySQL down more than 5 in 60 minutes
action=pipe '%s' /usr/bin/mail -s 'MySQL connection quality degradation' root
window=3600
thresh=5
```

ภาพที่ 3.11 แสดงตัวอย่างการกำหนด Rule ใน SEC

จากภาพที่ 3.11 เป็น Rule ที่ถูกสร้างขึ้นเพื่อตรวจสอบ Error หลังจากที่ผู้ใช้งานไม่สามารถทำการเชื่อมต่อไปยัง MySQL Server ได้ โดยจากการตรวจสอบข้อมูลล็อก จะมีการแจ้งเตือนผ่าน error.log คือ *mysql_connect(): Can't connect to local MySQL server through socket* ซึ่งหาก SEC มีการตรวจจับ Error ดังกล่าว จะมีการกำหนด Action ให้ทำการสร้าง Event

MySQL_DOWN จากนั้นจึงทำการตรวจสอบสถานะของ MySQL ว่ามีสถานะการใช้งานเป็น Down จริงหรือไม่ ผ่าน Script ดังภาพที่ 3.12

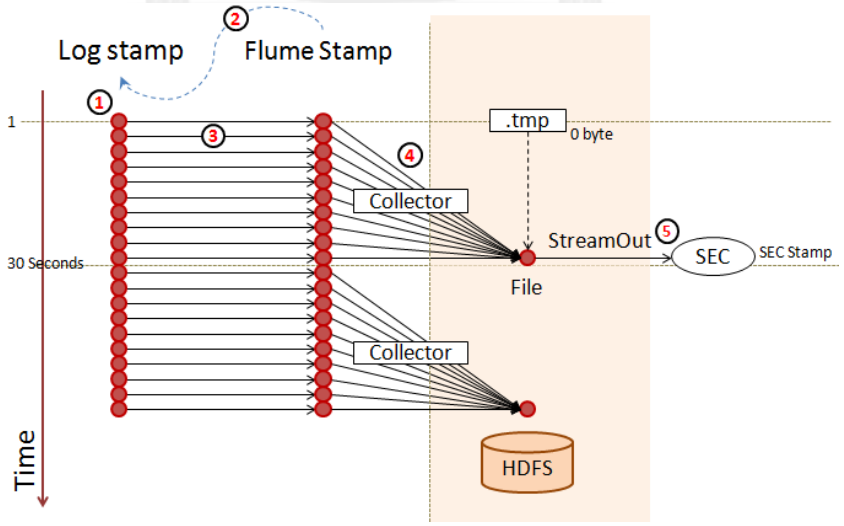
```
#!/bin/sh
# Shell command for Check Status MySQL
#
echo "Start of ERR0001-database-CheckStatus shell script"

if mysqladmin ping | grep -q "alive"; then
echo "MySQL is Up"
else
echo "MySQL is Down"
sudo /etc/init.d/mysql restart
fi

echo "End of shell script"
```

ภาพที่ 3.12 แสดง Script ตรวจสอบการทำงานของ MySQL

โดยหากตรวจสอบพบว่าสถานะของ MySQL ไม่สามารถใช้งานได้จริง จะมีการ Restart MySQL ใหม่ แต่หากไม่สามารถ start ระบบขึ้นมาใหม่ได้ หรือหาก Start ได้ แต่ยังคงเกิด Event:MySQL_DOWN ขึ้นอีกมากเกิน 5 ครั้งภายในเวลา 60 นาที เราอาจสันนิษฐานได้ว่าคุณภาพของการให้บริการในส่วนของ MySQL อาจไม่เสถียรหรือเกิดปัญหาขึ้นในระบบ จึงทำการส่ง e-mail แจ้งเตือนไปยังผู้ดูแลระบบเพื่อให้มาตรวจสอบอีกครั้งหนึ่ง ดังนั้นจะเห็นว่าหากมีใช้ Rule-based เพื่อเข้ามาช่วยในการตรวจสอบระบบ จะช่วยให้สามารถจัดการกับปัญหาได้อย่างอัตโนมัติหรือช่วยแก้ปัญหาได้เร็วยิ่งขึ้น

3.2 หลักการทำงานของ Flume และ SEC

หลักการทำงานของ SEC จะมีลักษณะการทำงานแบบ Stream event สำหรับการตรวจจับในลักษณะ Rule-based approach ซึ่งใช้วิธี Regular expression เพื่อทำการตรวจจับรูปแบบความผิดปกติของข้อมูล Log จากนั้นจึงทำการกำหนด Action ที่ต้องการ ซึ่งการทำงานร่วมกับ Flume นั้นจะมีหลักการเพื่อเขียนและตรวจจับความผิดปกติของข้อมูลจะมีลักษณะดังภาพที่ 3.13



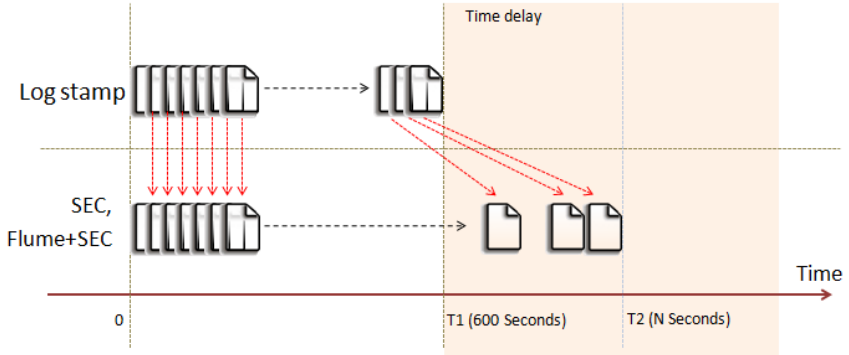
ภาพที่ 3.13 แสดงการทำงานของ Flume+SEC

ซึ่งการเก็บรวบรวมข้อมูล Log ของ Flume นั้น มีสถาปัตยกรรมดังที่ได้กล่าวไว้ในหัวข้อที่ 2.8 นั่นคือ จะมีการสร้างข้อมูล Log ส่งต่อไปยังส่วน Collector ซึ่งเป็นส่วนรวบรวมข้อมูล จากนั้นจึงทำการส่งต่อไปยัง HDFS เพื่อเก็บข้อมูล จึงทำให้มีขึ้นตอนการจำลองการ Feed Log ดังนี้

1. จำลองการ Feed ข้อมูลล็อกเป็นขนาดไฟล์ต่อวินาที (bps)
2. กำหนด Flume เพื่อให้ Stream ข้อมูลจากการ Feed ข้อมูลล็อกดังกล่าว
3. ส่งข้อมูลต่อไปยัง Flume collector เพื่อรวบรวมข้อมูลล็อก
4. Flume collector จะสร้างไฟล์ .tmp ขึ้นมาเพื่อรอรวบรวมข้อมูลแล้วเขียนต่อไปยัง HDFS (30 วินาที)
5. จากนั้นทำการ Stream ข้อมูลออกไปยัง SEC เพื่อทำการ Match หรือตรวจจับรูปแบบที่จะระบุถึงความผิดปกติของข้อมูล Log ที่เราต้องการ

3.3 การประเมินประสิทธิภาพการทำงาน

การวัดประสิทธิภาพการทำงานในที่นี้ จะทดสอบความสามารถในการเขียนข้อมูล log และตรวจจับรูปแบบเพื่อระบุความผิดปกติของข้อมูลที่ต้องการ โดยมีการออกแบบดังนี้



ภาพที่ 3.14 แสดงการทำงานในลักษณะ Real-time ของ SEC และ Flume+SEC

จากภาพที่ 3.14 แสดงถึง Baseline ในการวัดประสิทธิภาพเชิง Real-time ของ SEC และ Flume+SEC โดยจำลองการ Feed ข้อมูล Log ในแต่ละขนาดตั้งแต่ 1 Event – 24,000 Event ต่อวินาที เข้าไปในระบบเป็นเวลา 10 นาที จำนวน 10 ครั้ง จากนั้นจึงทำการวัดผลโดยคำนวนจากการหาค่าเฉลี่ยจากเวลาในการเกิด Delay ของ Event

3.4 สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือ

Data set

Source: แหล่งที่มาของข้อมูล Log มาจากเครื่อง DNS Server (161.200.80.1) และ Apache Web Server (161.200.80.71) ของทางคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์ในช่วงเดือนธันวาคม 2555

Source types: ข้อมูล Standard syslog

Example. *Nov 06 00:00:30 DnsServer2.eng.chula.ac.th Nov 6 00:00:30 DnsServer2 named[30942]: error (connection refused) resolving 'a1363.g.akamai.net/A/IN': 161.200.192.248#53*

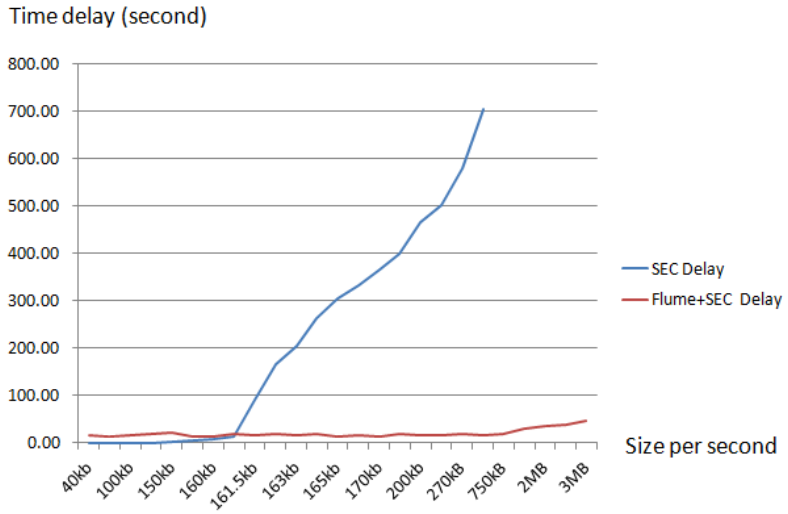
Record Size: ทดสอบข้อมูลตั้งแต่ 1 – 24,000 Events/second

Platform

- CPU: Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz Processors, RAM: 4 GB
- OS: Ubuntu 11.10 (GNU/Linux 3.0.0-26-server x86_64)

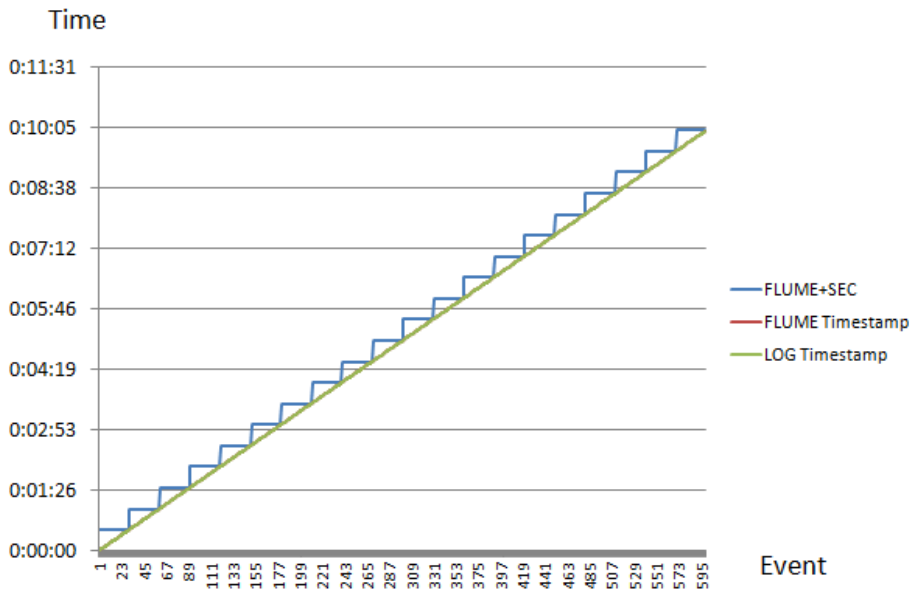
3.5 ผลการทดลองเบื้องต้น

จากการทดลองประสิทธิภาพการทำงานของ SEC และ Flume+SEC นั้น มีผลลัพธ์ดังภาพที่ 3.15 พบว่าทั้ง SEC และ Flume+SEC ไม่มีการตกหล่นในการเขียนและตรวจจับข้อมูล (Drop performance) และจากงานวิจัย [34] ได้นำ SEC มาใช้เพื่อทดสอบประสิทธิภาพความแม่นยำในการตรวจจับข้อมูล ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่า SEC สามารถตรวจจับข้อมูลได้ 100% โดยไม่มีการตกหล่น แต่จะเห็นว่าในช่วงขนาดของไฟล์ต่อวินาทีไม่เกิน 160kb นั้น ประสิทธิภาพของ SEC แทบจะไม่มีการ Delay เกิดขึ้น (Real-time) แต่เมื่อขนาดไฟล์ใหญ่เกินกว่า 160kb จะเริ่มเห็นการเกิด Delay อย่างชัดเจนขึ้นเรื่อยๆ ในขณะที่ Flume+SEC นั้น ยังคงมีประสิทธิภาพในการรับมือกับขนาดของข้อมูลที่เพิ่มขึ้นเรื่อยๆ ดังนั้นจะเห็นว่าการนำ Flume+SEC มาประยุกต์ใช้งาน ทำให้สามารถแก้ปัญหาในเรื่องขนาดของข้อมูลเชิง Big data เพื่อรองรับการทำงานแบบเรียลไทม์ได้ดียิ่งขึ้น



ภาพที่ 3.15 เปรียบเทียบประสิทธิภาพการทำงานระหว่าง SEC และ Flume+SEC

เมื่อวิเคราะห์ประสิทธิภาพของการทำงานในส่วนของ Flume+SEC นั้น จะพบว่าสาเหตุที่ทำให้เกิด Delay ในการตรวจจับข้อมูลจะมีลักษณะดังรูปที่ 3.16



ภาพที่ 3.16 แสดงประสิทธิภาพการทำงานของ Flume+SEC

จากภาพที่ 3.16 ได้ทำการทดสอบข้อมูล Log ขนาด 200kb/sec จะเห็นว่าประสิทธิภาพการทำงานของ Flume+SEC มีผลลัพธ์ดังกราฟ เนื่องจากการทำงานของ Flume collector จะทำการสร้างไฟล์ .tmp (0 byte) จากนั้นจึงรอเป็นเวลา 30 วินาที เพื่อรวมไฟล์และเขียนไฟล์ใหม่ต่อไปยัง HDFS แล้วจึงสามารถอ่านและ Detect ข้อมูลได้ ซึ่งจะทำให้เกิดช่วง Delay ของการ Detect ข้อมูล ทำให้ได้ผลลัพธ์ดังที่แสดงในภาพที่ 3.16

บทที่ 4
การพัฒนาระบบตามข้อจำกัดทางเวลาข้อมูล

จากผลการทดลองในบทที่ 3 พบว่าการตรวจสอบความน่าเชื่อถือโดยการใช้ Rule-based เข้ามาช่วยในการตรวจสอบระบบ จะช่วยให้จัดการกับปัญหาได้เร็วยิ่งขึ้นจริง แต่อาจจะล่าช้าในการ Detect เนื่องจาก การจะรองรับปริมาณข้อมูลจำนวนมากในรูปแบบเรียลไทม์นั้น อาจต้องคำนึงถึงปัญหาในส่วนของ Input rate และความซับซ้อนที่เกิดขึ้นภายในระบบ ดังนั้น ทางผู้วิจัยจึงคำนึงถึงความเป็นเรียลไทม์ในการตรวจจับซึ่งการทำงานของระบบตรวจสอบอาจจะต้องมีการกำหนดคุณภาพการให้บริการ (Quality of Service) ในการตรวจสอบ สิ่งผิดปกติหรือ Error ที่เกิดขึ้นภายในระบบตามระยะเวลาที่กำหนด (Deadline) เพื่อตรวจสอบความน่าเชื่อถือของระบบด้วยการวิเคราะห์จากข้อมูล Log ดังนั้น ปัญหาหลักสำหรับผู้ดูแลระบบคือทำอย่างไรให้ผู้ใช้บริการมั่นใจได้ว่าการให้บริการนั้นจะเป็นไปตาม SLA หรือตามข้อตกลงของระดับการให้บริการที่ได้ให้ไว้ ซึ่งในงานวิจัยนี้ได้ออกแบบโมเดลเพื่อมุ่งเน้นไปที่การกำหนดข้อจำกัดของเวลา (Timing Constraint Event Model) ที่เหมาะสม รวมทั้งหาความน่าจะเป็นหรือโอกาสที่ระบบตรวจสอบจะตรวจจับเกินระยะเวลา ที่ตั้งไว้

4.1 Timing Constraint Event Model

4.1.1 ข้อกำหนดของ Event (Event specification)

ในงานวิจัยนี้ได้ทำการนิยามคำว่า Event เป็นการตรวจสอบสิ่งที่เรากำลัง Monitor ซึ่งแต่ละ Event จะแสดงถึงสิ่งที่เกิดขึ้นในระบบ โดยข้อมูลเกี่ยวกับ event จะประกอบด้วยข้อมูลต่างๆ เช่น Timestamp เป็นส่วนบอกถึงเวลาที่ Event เกิดขึ้น, ข้อมูล IP address หรือชื่อของ node ที่ generate log และ Text message เป็นข้อมูลเกี่ยวกับ event ที่บ่งบอกถึงเหตุการณ์ปกติ (regular) ไปจนถึง Error ที่เกิดขึ้นในช่วงที่ระบบมีการทำงาน [16] ดังนั้น Timestamp สามารถสื่อถึงความสัมพันธ์ระหว่าง Event ที่เกิดขึ้นในระบบได้

งานวิจัยนี้นิยาม Event ที่เกิดขึ้น เป็น @- function ซึ่งแสดงถึงความสัมพันธ์ระหว่าง Event ที่ถูกสร้างขึ้นระหว่างการให้บริการของระบบกับครั้งที่ที่ Event เกิด

<u>บทนิยาม 1</u> กำหนด Event log และ integer i เป็นจำนวนเต็มบวก $i \in N^+$, จะได้

$$@(log, i) = occurrence\ time\ of\ the\ ith\ instance\ of\ event\ log \quad (4.1)$$

โดยที่ i เป็นพารามิเตอร์ของ @-function

$$\forall event\ log, \forall i \in N^+, @(log, i) < @(log, i + 1) \quad (4.2)$$

4.1.2 ข้อกำหนดของ Monitoring Timing constraint specification

ในงานวิจัยนี้เรากำหนด @-function บน Framework ของระบบ Monitoring โดยแบ่งลักษณะ Event ที่เกิดขึ้นออกเป็น 3 ลักษณะ คือ

1. **@(actual, i):** Event เกิดขึ้นจริงภายใต้อุปกรณ์ (Hardware) หรือซอฟต์แวร์ (Software)
2. **@(log, i):** Event ที่อุปกรณ์ (Hardware) หรือซอฟต์แวร์ (Software) ลงรายละเอียด Timestamp หรือข้อมูล log
3. **@(detect, i):** Event ที่ระบบ Monitoring ตรวจสอบได้

โดยตั้งสมมติฐานว่าเวลาที่ Actual event เกิดขึ้นกับเวลาที่มีการลง Log event นั้น ไม่แตกต่างกัน นั่นคือ

$$@(\text{actual}, i) = @(\text{log}, i) \tag{4.3}$$

โดย simple constraint [35] สามารถอธิบายได้จาก $T_1 \leq T_2 + D$, โดยที่ D เป็นจำนวนเต็มบวก ดังนั้นจึงสามารถประยุกต์เป็นบทนิยาม 2

บทนิยาม 2 Deadline constraint สามารถนิยามได้เป็น

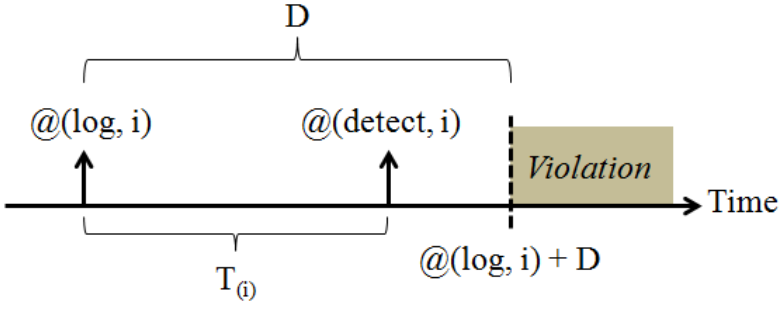
$$\{ @(\text{detect}, i) \leq @(\text{log}, i) + D \} \tag{4.4}$$

โดยที่ $D > 0$ และ i เป็นจำนวนครั้งที่เกิด Event

ตัวอย่างที่ 1 สมการต่อไปนี้จะเป็นการแสดง Deadline constraint ที่กำหนดให้ทุกๆ Event จะถูก Detect ภายใน 15 วินาที

$$@(\text{detect}, i) \leq @(\text{log}, i) + 15$$

ในกรณีนี้จะเกิดการละเมิด (Violation) ต่อเมื่อการตรวจจับ (Detection) เกินระยะเวลาที่กำหนด เช่น จากตัวอย่างที่ 1 เมื่อกำหนด Deadline ไว้ที่ 15 วินาที หากระบบ Monitoring ใช้เวลาในการตรวจจับข้อมูล log มากเกินกว่า 15 วินาทีจะถือว่าเกิดการละเมิด (Violation) นั้นคือไม่สามารถ Detect Event ได้ภายในระยะเวลาที่กำหนดไว้ นั่นเอง



ภาพที่ 4.1 แสดงถึงความไม่แน่นอนในการ Detect

<u>บทนิยาม 3</u> จากบทนิยามที่ 2 คือ @(detect, i) จะต้องถูก Detect ภายในระยะเวลา @(log, i) + D ดังรูปที่ 4.1 ดังนั้น จะเห็นว่า เวลาที่ระบบ Monitoring จะ Detect ได้นั้นไม่แน่นอน อาจจะมีสิทธิ์มากกว่าหรือน้อยกว่า @(log, i) + D ก็ได้ ดังนั้นเราจึงนิยามได้ว่า Detection time เป็นตัวแปรสุ่ม (Random variable), $T_{(i)}$,โดยที่

$$T(i) = @(detect, i) - @(log, i) \quad (4.5)$$

จะเห็นว่าหาก $T_{(i)} \leq D$ จะทำให้ระบบ Monitoring สามารถการันตีได้ว่า Event ที่เกิดขึ้นในระบบนั้นสามารถตรวจจับได้ภายในระยะเวลา Deadline ที่กำหนด ในขณะที่ หาก $T_{(i)} > D$ จะถือว่าเกิดการ Violation ภายใต้ Deadline constraint ที่กำหนด

4.1.3 ข้อกำหนดของ Monitoring Probabilistic Constraint Specification

จากบทนิยามที่ 3 ว่าระบบ Monitoring จะการันตี Service Level agreement ต่อเมื่อ $T_{(i)} \leq D$ แต่เนื่องจากที่เรากำหนดค่า $T_{(i)}$ เป็น Random variable และ $T_{(i)} \leq D$ นั้นไม่แน่นอน ทำให้เราสามารถกำหนดรูปแบบให้เป็นลักษณะของโมเดลความน่าจะเป็น (Probabilistic model) เพื่อให้สามารถคาดเดาและคำนวณโอกาสที่ระบบ Monitoring จะสามารถ Detect ได้ภายในระยะเวลาที่กำหนด (Deadline constraint) ภายใต้ข้อตกลงของการให้บริการหรือไม่ และจากที่กล่าวไปในข้างต้น คือ ในงานวิจัยนี้มองถึงระบบที่มีลักษณะเป็น Soft real-time ซึ่งลักษณะของ Soft real-time คือ ผลลัพธ์ที่ได้รับหลังจากระยะเวลาที่กำหนด (Deadline) ยังคงสามารถนำมาใช้งานได้ หากอยู่ภายใต้ข้อตกลงในการให้บริการ [36]. ซึ่งในระบบที่มีลักษณะเป็น Soft real-time นั้น Probabilistic deadline [36, 37] มักจะถูกนำมาใช้เป็นตัววัดสำหรับการันตี SLA ซึ่งเราสามารถนำ Probabilistic model มาประยุกต์ใช้สำหรับตรวจสอบความน่าจะเป็นว่า Event จะสามารถ detect ได้ภายในขอบเขตที่ Soft real-time ถูกกำหนดหรือไม่ เช่น “90% ของการ Monitoring จะต้องสามารถตรวจจับ Error จาก Event log ไดภายใน 10 วินาที” เป็นต้น

<u>บทนิยาม 4</u> ประยุกต์ใช้ Probabilistic deadline ในการคำนวณโอกาสความน่าจะเป็นที่ระบบ Monitoring จะสามารถ Detect event ภายใต้ Deadline ของ SLA ที่กำหนด โดยที่สมมติค่า $T_{(i)}$ เป็น Normal distribution

$$P\{SLA(D)\} = P\{T \leq D\} \quad (4.6)$$

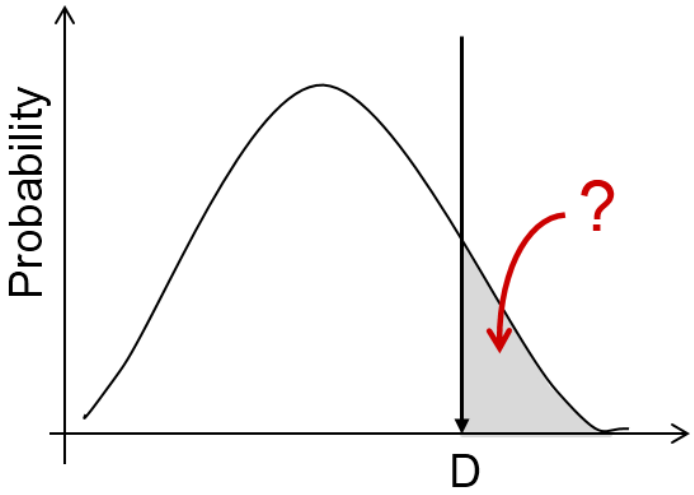
โดยที่

- T คือ sample ของ Detection time; $T_{(i)}$ = @(detect, i) - @(log, i)
- ค่าโอกาสการเกิด (Probability Value) ที่บ่งบอกโอกาสในการเกิดการละเมิดข้อตกลงของการให้บริการ (SLA) ใช้สัญญลักษณ์ คือ (P) โดย P{SLA(D)} คือความน่าจะเป็นที่การ Detect จะทำได้ภายในระยะเวลา Deadline (D)
- ค่าเฉลี่ยของตัวอย่าง (Sample mean) เขียนแทนด้วย Mean ($\overline{X}$)

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{4.7}$$

- ความคลาดเคลื่อนมาตรฐาน (Standard deviation)

$$S = \sqrt{\frac{1}{N} \sum_{i=1}^{n} (X_i - \bar{X})^2} \tag{4.8}$$



ภาพที่ 4.2 แสดงความน่าจะเป็น Miss deadline ratio

จากภาพ 4.2 จากที่เราสมมติค่า T(i) เป็น Normal distribution หากต้องการคำนวณหาโอกาสที่จะ Detect เกินเวลา Deadline ที่เรากำหนดไว้หรือไม่ จะสามารถคำนวณค่าความน่าจะเป็นได้จากพื้นที่ฝั่งขวาของกราฟ ตามทฤษฎีของ Normal distribution ดังนั้น จากสมการที่กล่าวไว้ข้างต้น เมื่อเรากำหนด Deadline เราจะสามารถคำนวณโอกาสที่จะ Detect เกินเวลา Deadlineได้ดังสมการ

$$P\{T_{(i)} > D\} = 1 - P\{SLA(D)\} \tag{4.9}$$

จาก $T_{(i)}$ เป็น Normal distribution ดังนั้นจะสามารถคำนวณสูตรด้วย Z-Score

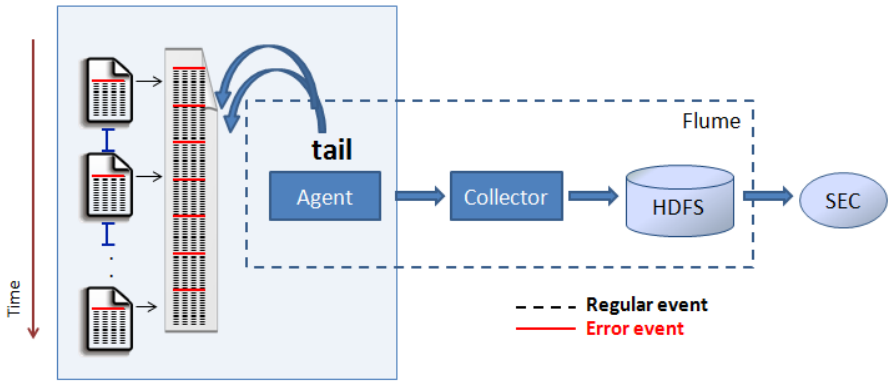
- Z-score

$$Z = \frac{X_i - \bar{X}}{S} \tag{4.10}$$

จะเห็นว่างานวิจัยนี้จะนำแนวคิดของทฤษฎีทางด้านสถิติมาประยุกต์และคาดการณ์ถึงสิ่งที่อาจเกิดขึ้น โดยนำทฤษฎีการกระจายค่าความน่าจะเป็น (Probability Distribution) มาคำนวนถึงโอกาสที่จะเกิดการละเมิดข้อตกลงของการให้บริการและเวลาในการกำหนด Deadline ของ SLA ที่ควรจะเป็น

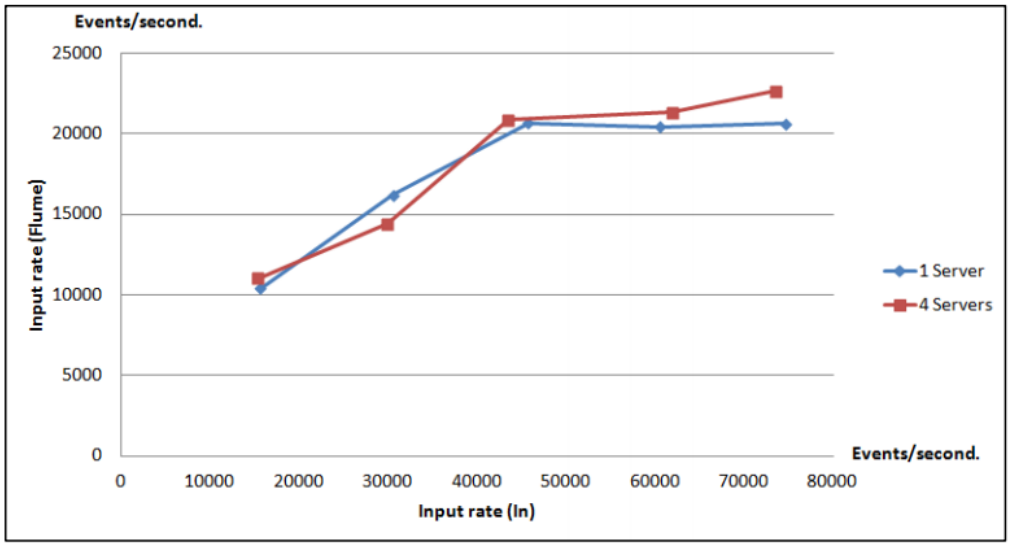
4.2 การประเมินผลกระทบของ Input rate กับการทำงานของ Flume

จากการทดลองเบื้องต้นในบทที่ 3 จะเห็นได้ว่าระบบตรวจสอบที่ใช้นั้นมีประสิทธิภาพความแม่นยำในการตรวจจับข้อมูล ซึ่งผลลัพธ์ที่ได้แสดงให้เห็นว่า SEC สามารถตรวจจับข้อมูลได้ 100% โดยไม่มีการตกหล่น แต่จะเห็นว่าเวลาที่ใช้ในการ Detect ยังไม่มีประสิทธิภาพรวมถึงขนาดของข้อมูลล็อก ที่ใช้ในการทดลองนั้นยังมากเท่าที่ควร ดังนั้น ทางผู้วิจัยจึงทำการทดสอบประสิทธิภาพการทำงานของ Flume โดยในการประเมินประสิทธิภาพการทำงานของ Flume นั้น จะทดสอบความสามารถในการเขียนข้อมูล log และตรวจจับรูปแบบเพื่อระบุความผิดปกติของข้อมูลที่ต้องการ โดยมีการออกแบบดังภาพที่ 4.3



ภาพที่ 4.3 แสดงแบบจำลองการวัดประสิทธิภาพของ Flume

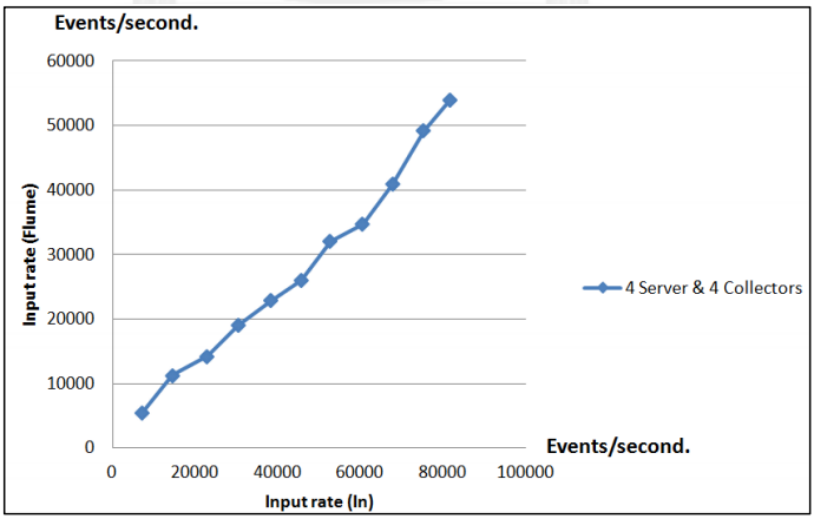
จากภาพที่ 4.3 แสดงถึงการจำลองการ Feed ข้อมูล Log โดยกำหนดให้ Agent อ่านข้อมูลจาก Log แล้วทำการส่งต่อไปยังส่วน Collector และ Storage (HDFS) จากนั้นทำการตรวจจับข้อมูลที่เป็น Error event จาก SEC ซึ่งจะทำการประเมินประสิทธิภาพจากการคำนวนถึง Input rate ในฝั่งขาเข้าและฝั่งขาออก โดยทำการทดลองทั้ง 1 Agent และ 4 Agents เพื่อเปรียบเทียบความแตกต่างในส่วนของ Input rate ซึ่งได้ผลลัพธ์ดังภาพที่ 4.6



ภาพที่ 4.4 แสดง Input rate เปรียบเทียบระหว่าง 1 Agent และ 4 Agents

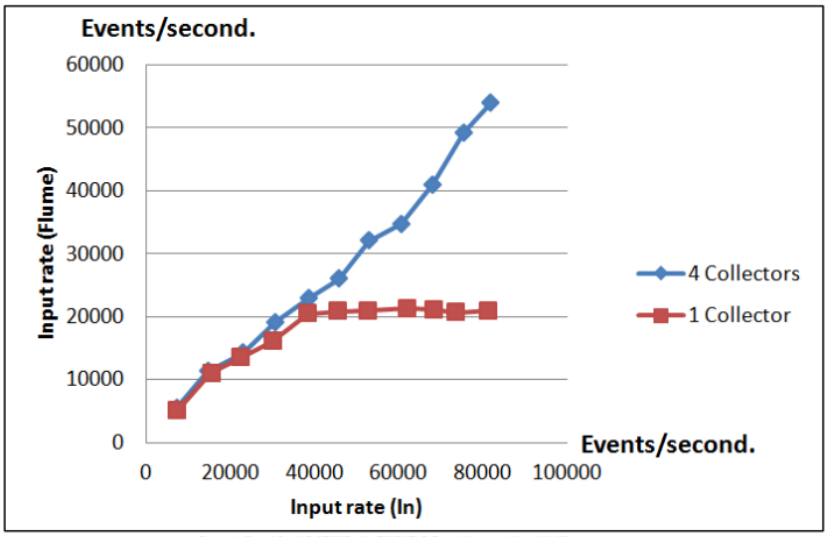
จากภาพที่ 4.4 แสดงการจำลองความเร็วในการส่งข้อมูลต่อ 1 Collector โดยเพิ่มจำนวน Event มากขึ้นเรื่อยๆ จะเห็นว่าจำนวน Agent ที่เพิ่มมากขึ้น ไม่ได้ส่งผลให้ Input rate ในฝั่งของ Flume เปลี่ยนไป แต่ในขณะเดียวกัน จะเห็นว่าขอบเขตการทำงานของ Flume ต่อ 1 Collector สามารถเขียนและรองรับการทำงานได้เต็มที่ไม่เกิน 20,000 Events/second หากเกินกว่านี้ อาจเกิดการล่าช้าในการเขียนไฟล์ลง HDFS

เมื่อเราคำนึงถึงความเร็วในการข้อมูลขนาดใหญ่นั้น การจะรับมือกับ Input rate ที่เกิดขึ้นในความเป็นจริง เราอาจไม่จำเป็นที่จะใช้ Collector เพียงตัวเดียวเพื่อรองรับข้อมูลทั้งระบบ ดังนั้น ในงานวิจัยนี้จึงได้ทดลองประสิทธิภาพการทำงานเมื่อใช้ Collector มากกว่า 1 ตัว โดยผลการทดลองแสดงดังภาพ



ภาพที่ 4.5 แสดงผลการทดลอง 4 Agents และ 4 Collectors

จากภาพที่ 4.5 แสดงการจำลองความเร็วในการส่งข้อมูลโดยเพิ่ม Input rate ทุกๆ 8,000 Events/second. ต่อ 4 Collectors จะเห็นว่าสามารถรองรับข้อมูลที่เพิ่มมากขึ้นได้ดีกว่าการใช้ Collector เพียงตัวเดียว กล่าวคือ หากเราทราบว่าข้อมูลภายในระบบจะมี Input rate เพิ่มมากขึ้น เราก็สามารถรองรับการขยายของระบบได้โดยการเพิ่ม Collector ดังแสดงในภาพที่ 4.6



ภาพที่ 4.6 แสดงผลการทดลองเปรียบเทียบระหว่าง 1 Collector และ 4 Collectors

จากภาพที่ 4.6 แสดงให้เห็นว่าเราสามารถเพิ่ม Collector ในโครงสร้างของระบบ Monitoring เพื่อรองรับปริมาณข้อมูลที่เพิ่มมากขึ้นได้ และเมื่อเราทราบถึง Input rate ที่เหมาะสมสำหรับ Collector 1 ตัว หากเราต้องการให้ตรวจสอบความผิดพลาดให้ได้ภายในระยะเวลาที่กำหนด (Deadline) ตาม SLA อาจต้องทำการเพิ่มโครงสร้างในระบบตรวจสอบเพื่อเพิ่มประสิทธิภาพในการตรวจจับให้ได้ภายในระยะเวลาที่กำหนด

4.3 การออกแบบโครงสร้างเพื่อเพิ่มประสิทธิภาพในการตรวจจับ

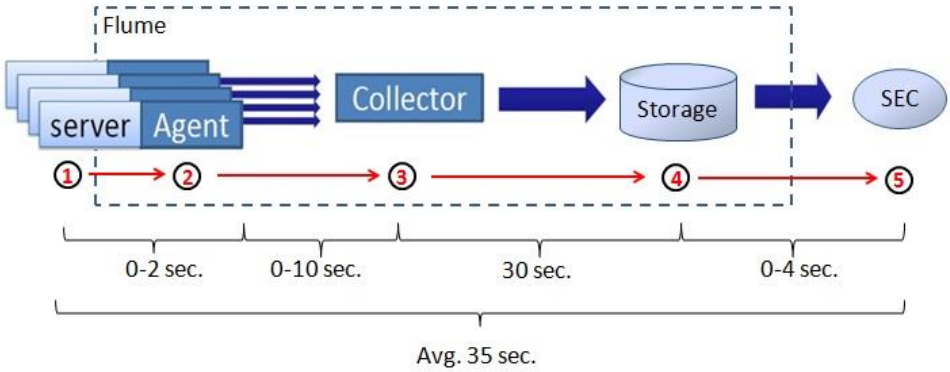
เมื่อเราทำการวิเคราะห์เวลาในการไหลของข้อมูลแต่ละส่วนของโครงสร้างระบบ Monitoring ในแต่ละช่วงของ Input rate ให้ผลจากการทดลองดังตาราง ที่ 4.1

ตารางที่ 4.1 แสดงเวลาในการไหลของข้อมูลแต่ละส่วนของโครงสร้างระบบ

Input rate	Server To Agent	Agent To Collector	Collector To Storage	Storage To SEC	Average
8,000 Events/s.	0-2s.	0-10s.	30s.	0-4s.	35s.
16,000 Events/s.	1-2s.	0-13s.	30s.	0-5s.	37s.
24,000 Events/s.	1-2s.	0-30s.	30s.	1-30s.	57s.

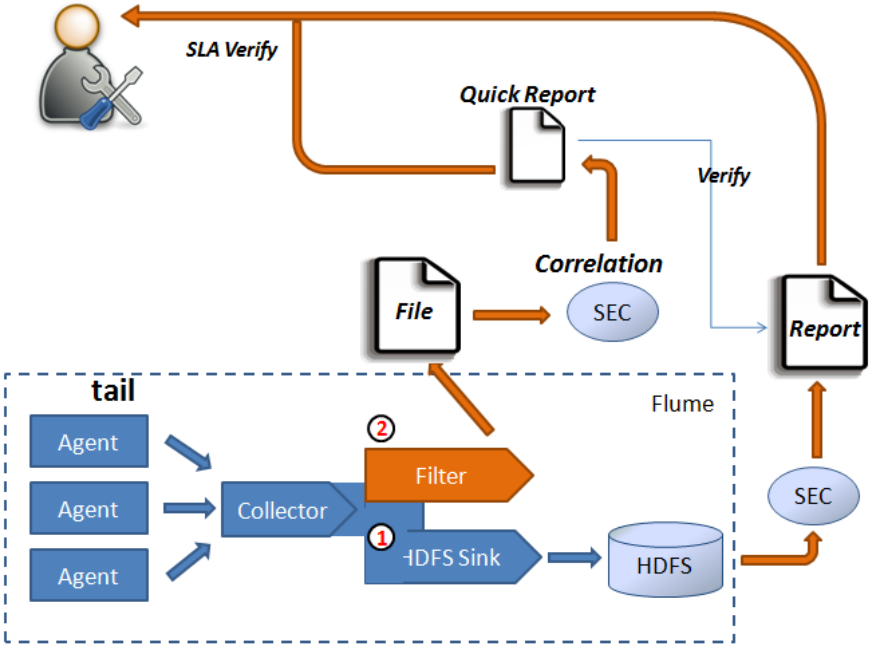
จากตัวอย่าง Input rate เท่ากับ 8,000 Events/s พบว่า ข้อมูลที่ไหลเกิดจาก Server มาบันทึกยัง Flume agent มีเวลาเฉลี่ยอยู่ที่ประมาณ 0-2 วินาที จากนั้นจึงส่งข้อมูลต่อไปยัง

Collector ซึ่งใช้เวลาประมาณ 0-10 วินาที จากนั้น Flume collector จะสร้างไฟล์ .tmp ขึ้นมาเพื่อรอรวบรวมข้อมูลแล้วเขียนต่อไปยัง HDFS (30 วินาที) จากนั้น SEC จะทำการตรวจจับข้อมูลที่เรากำหนดตาม Rule ซึ่งใช้เวลาประมาณ 0-4 วินาที ดังนั้นจะเห็นว่าข้อมูลที่ไหลจากต้นทางจนถึงปลายทางจะใช้เวลาในการ Detect ประมาณ 35 วินาที จากภาพที่ 4.7



ภาพที่ 4.7 แสดงผลการวิเคราะห์เวลาในการไหลของข้อมูลแต่ละส่วน

จากภาพที่ 4.7 จะเห็นว่าส่วนที่ทำให้ระบบตรวจจับข้อมูลได้ช้าเนื่องจาก Collector จะทำการรอรวบรวมข้อมูลที่ไหลมาจากส่วนต่างๆของ Agent เพื่อเขียนไปยัง HDFS แต่หากกรณีที่เราต้องการตรวจสอบความผิดพลาดให้ได้ภายในระยะเวลาที่กำหนด (Deadline) ตาม SLA อาจต้องทำการเพิ่มโครงสร้าง Architecture เพื่อเพิ่มความสามารถในการตรวจจับ โดยมีการแก้ไขโครงสร้างการทำงานของ Flume เพิ่มเติมดังภาพที่ 4.8

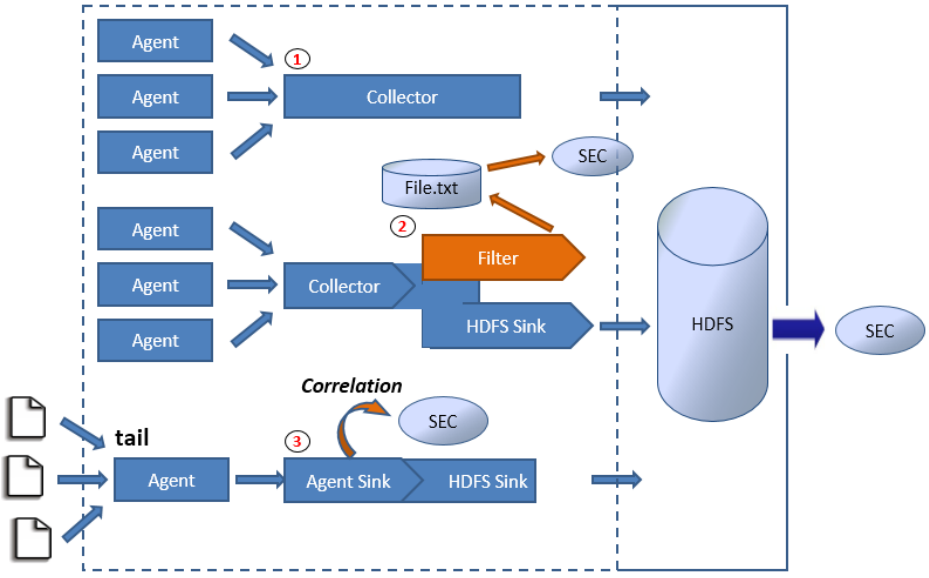


ภาพที่ 4.8 แสดงโครงสร้างการตรวจจับความผิดพลาดของระบบภายใต้ SLA ที่กำหนด

เพื่อให้ระบบสามารถตรวจความผิดปกติของข้อมูลล็อกได้ภายในระยะเวลาที่รวดเร็ว ทางผู้วิจัยจึงได้ทำการเพิ่มโครงสร้างในส่วนของ Filter เข้ามาใช้ โดยในการทำงานจะมีการกระจาย Collector Sink ออกเป็น 2 ทางคือ

1. ส่งต่อไปยัง HDFS เพื่อจัดเก็บ Raw log ทั้งหมดโดยไม่มีการผ่านการกรองหรือ Correlation เพื่อให้สามารถตรวจสอบและวิเคราะห์ข้อมูลในภายหลัง
2. เพิ่มประสิทธิภาพของระบบให้มีความเป็น Real-time ในการตรวจจับความผิดปกติของข้อมูล Log มากยิ่งขึ้น โดยมีการส่งไปยังส่วนที่เรียกว่า Filter เพื่อทำการคัดกรองข้อมูลเฉพาะส่วนที่เกี่ยวข้องกับการเกิดข้อผิดพลาดในระบบ และนำผลที่ได้นั้นไป Correlation เพื่อวิเคราะห์ถึงสาเหตุที่ทำให้ระบบเกิดข้อผิดพลาด

จะเห็นว่าองค์ประกอบสำคัญของ Soft real-time monitoring ในงานวิจัยนี้คือความสามารถที่ระบบจะตรวจสอบความผิดพลาดได้ตามข้อจำกัดทางเวลา หรือภายในระยะเวลาที่กำหนด (Deadline) ตาม SLA โดยแต่ละระบบอาจมีความต้องการความต้องการในส่วนของ SLA ในระดับที่ต่างกัน โดยอาจเป็น Soft หรือ Hard real-time ขึ้นอยู่กับว่าระบบนั้นๆจะมีผลกระทบต่อการละเมิด deadline ได้มากน้อยเพียงใด หากผลลัพธ์ที่ได้รับหลังจากระยะเวลาที่กำหนดยังคงสามารถนำมาใช้งานได้ แต่อาจส่งผลให้ละเมิดข้อตกลงในการให้บริการ เราจะเรียกว่าเป็น Soft ในขณะที่ Hard หมายถึงไม่สามารถละเมิดข้อตกลงของการให้บริการโดยเด็ดขาด ซึ่งในสภาพแวดล้อมแต่ละระบบ เราสามารถจัดโครงสร้าง Monitoring Framework ได้หลายรูปแบบขึ้นอยู่กับตามความต้องการของระบบนั้นๆ ตัวอย่าง โครงสร้าง Monitoring Framework แสดงอยู่ในภาพที่ 4.9



ภาพที่ 4.9 แสดงการออกแบบโครงสร้างเปรียบเทียบกับข้อจำกัดของเวลา

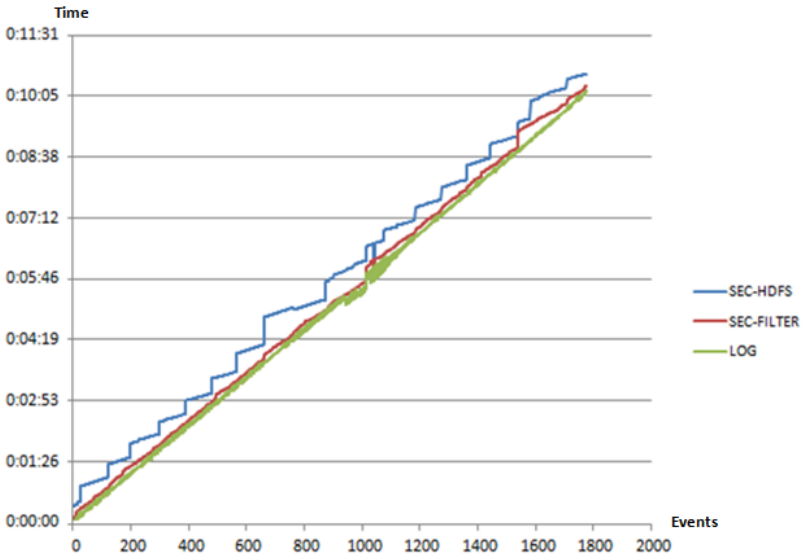
เมื่อตัวชี้วัดของระบบ Real-time Monitoring คือ เวลาในการ Detect ให้ได้ตาม SLA ดังนั้นการออกแบบโครงสร้างการทำงานอาจจะสามารถทำได้หลายแบบขึ้นอยู่กับสถานการณ์ในแต่ละ

ระบบ โดยไม่จำเป็นจะต้องมีโครงสร้างการทำงานของระบบตรวจสอบเพียงแบบใดแบบหนึ่ง เช่น การทำงานของระบบๆหนึ่ง อาจมีลักษณะโครงสร้างการตรวจจับ ดังภาพที่ 4.8

1. ในบางสถานการณ์ หากไม่ได้มีการกำหนดระยะเวลาในการตรวจจับ เราอาจใช้โครงสร้างการทำงานของ Flume แบบดั้งเดิมเพื่อทำการหาความสัมพันธ์ข้อมูล หลังจากที่ข้อมูลถูกส่งต่อจากต้นทางมายัง HDFS ซึ่งข้อดีของระบบตรวจสอบ ในลักษณะนี้คือ ง่ายต่อการติดตั้งระบบ และมองเห็นภาพรวมในการหาความสัมพันธ์ของข้อมูลในลักษณะของการทำการงานในระบบแบบกระจาย แต่ข้อเสียคือ ผลลัพธ์ของการตรวจจับ อาจมีความล่าช้าในการตรวจจับสิ่งผิดปกติที่เกิดขึ้นในระบบ ซึ่งจากผลการทดลองสรุปได้ว่าเวลาเฉลี่ยในการตรวจจับ ของ Flume ตั้งแต่ต้นทางส่งต่อมายัง SEC ที่ความเร็วในการส่งข้อมูลไม่เกินความสามารถของโครงสร้างที่รับได้นั้นอยู่ที่ประมาณ 30-60 วินาที
2. หากภายในระบบบางส่วนต้องการลดระยะเวลาในการตรวจจับให้น้อยลงอาจจะใช้โครงสร้างในแบบที่ 2 โดยจะทำการกรองข้อมูลส่วนที่ต้องการ แล้วส่งต่อไปยัง SEC เพื่อทำการหาความสัมพันธ์อย่างรวดเร็ว ซึ่งโครงสร้างในลักษณะนี้จะช่วยให้ระบบแสดงผลลัพธ์และวิเคราะห์ข้อมูลได้เร็ว และยังคงสามารถเชื่อมโยงความสัมพันธ์ในส่วนต่างๆที่ส่งมายัง Collector ที่เรากำหนดไว้ได้ โดยผลจากการทดลองเมื่อกำหนดความเร็วในการส่งข้อมูลที่เหมาะสมในระบบ จะได้เวลาเฉลี่ยในการตรวจจับข้อมูลอยู่ที่ประมาณ 5-10 วินาที
3. หากต้องการตรวจจับและหาความสัมพันธ์ข้อมูลเฉพาะข้อมูลล็อกที่เกิดขึ้นภายใน Agent เราอาจจะใช้โครงสร้างในแบบที่ 3 ซึ่งในสถานการณ์นี้จะเหมาะสำหรับระบบที่ต้องมีความเป็นเรียลไทม์มากๆ โดยไม่จำเป็นต้องผ่านการหาความสัมพันธ์ในโหนด อื่นๆ กล่าวคือหากใช้โครงสร้างการตรวจจับในลักษณะนี้ อาจไม่ได้ตอบโจทย์ในลักษณะการทำงานของระบบที่เป็นแบบกระจายที่แท้จริง ซึ่งผลจากการทดลองเมื่อกำหนดความเร็วในการส่งข้อมูลที่ไม่เกิน Input rate ที่เหมาะสม จะใช้เวลาเฉลี่ยในการตรวจจับข้อมูลผิดปกติเพียง 0-3 วินาที

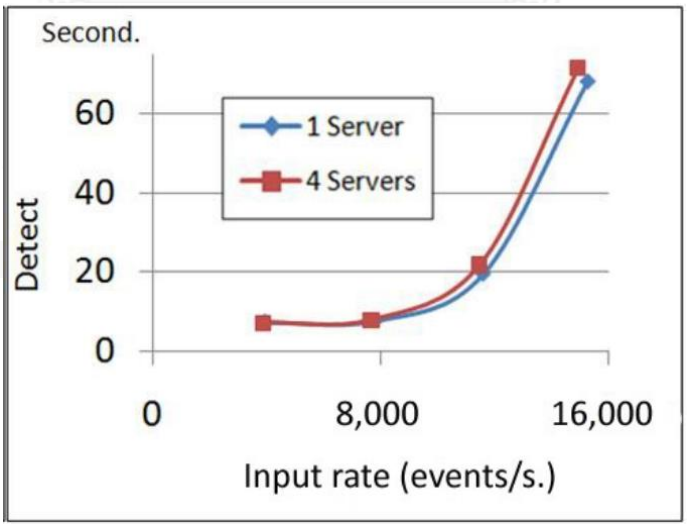
จะเห็นว่าโครงสร้างในแต่ละรูปแบบนั้นมีข้อดีและข้อเสียแตกต่างกัน ซึ่งในความเป็นจริงนั้นในระบบเพียงระบบเดียว สามารถออกแบบและกำหนดโครงสร้างเพื่อตรวจจับสิ่งผิดปกติในแต่ละส่วนตามระยะเวลาที่กำหนดได้หลากหลายรูปแบบ รวมทั้งสามารถใช้วิธีการวิเคราะห์ล็อกเชิงกฎเกณฑ์แบบกระจายเพื่อเพิ่มประสิทธิภาพให้มากขึ้นได้

หลังจากที่เราทำการแก้ไขโครงสร้างการทำงานของ Flume ดังภาพ 4.7 ซึ่งเมื่อทำการประเมินประสิทธิภาพของเวลาที่ใช้ในการตรวจจับข้อผิดพลาด ได้ผลลัพธ์ดังนี้



ภาพที่ 4.10 แสดงประสิทธิภาพในการตรวจจับ Error

จากภาพที่ 4.10 แสดงการทดสอบประสิทธิภาพในการตรวจจับข้อผิดพลาด โดยในการทดลองได้ทำการจำลองข้อมูลล็อก และทำการส่งข้อมูลจาก 3 Agents เป็นเวลา 10 นาที จากนั้นทำการหาเวลาเฉลี่ยในการตรวจจับ ซึ่งหากทำผ่าน Filter จะได้เวลาเฉลี่ยประมาณ 7-8 วินาที ในขณะที่เวลาเฉลี่ยในการตรวจจับผ่าน HDFS (โครงสร้างการทำงาน Flume แบบดั้งดิม) เท่ากับ 35 วินาที ดังนั้นจะเห็นว่าผลลัพธ์ที่ได้จากการตรวจจับแบบผ่านการ Filter (SEC-Filter) นั้นมีประสิทธิภาพในการตรวจจับมากกว่าการตรวจจับผ่าน HDFS ซึ่งจะช่วยให้สามารถตรวจจับข้อมูลและแก้ปัญหาได้เร็ว อีกทั้งยังอาจช่วยให้ไม่ส่งผลต่อการละเมิด SLA Deadline อีกด้วย



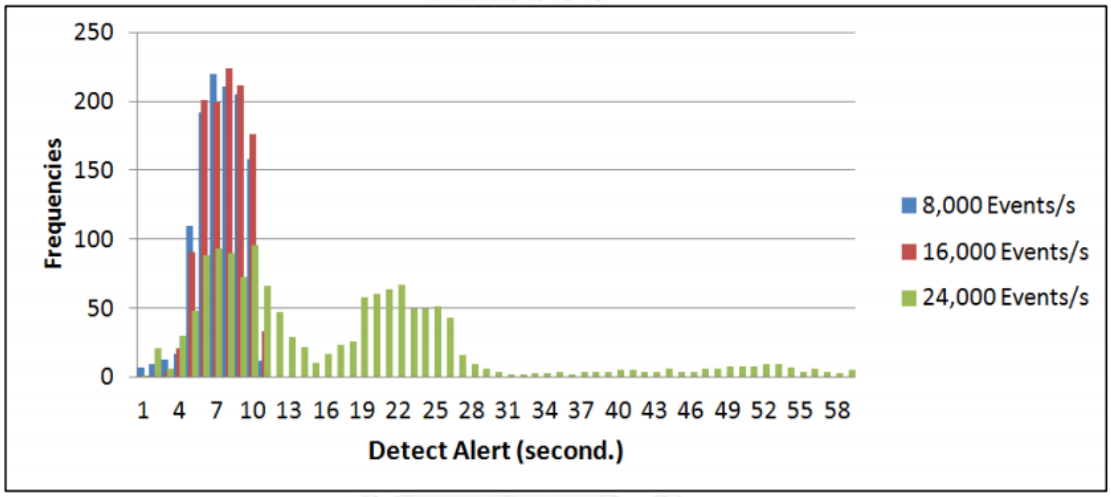
ภาพที่ 4.11 แสดงการตรวจจับ (Event Detection delay)

จากภาพ 4.11 แสดงถึงเวลาเฉลี่ยนในการ Detect event ซึ่งจะเห็นว่า หากระบบเรามีการกำหนด Deadline constraint ไว้ที่ 10 วินาที ในกรณีที่มี Input rate ไม่เกิน 20,000 Event/s.

ระบบจะยังมีโอกาสที่จะตรวจจับได้ภายใน 10 วินาที แต่หาก input rate สูงจนเกินความสามารถที่ระบบจะรับได้ จะทำให้เวลาที่ใช้ในการตรวจจับเพิ่มขึ้นอย่างมาก

4.4 ผลการทดลองการคำนวณหาความน่าจะเป็น Miss Deadline ratio

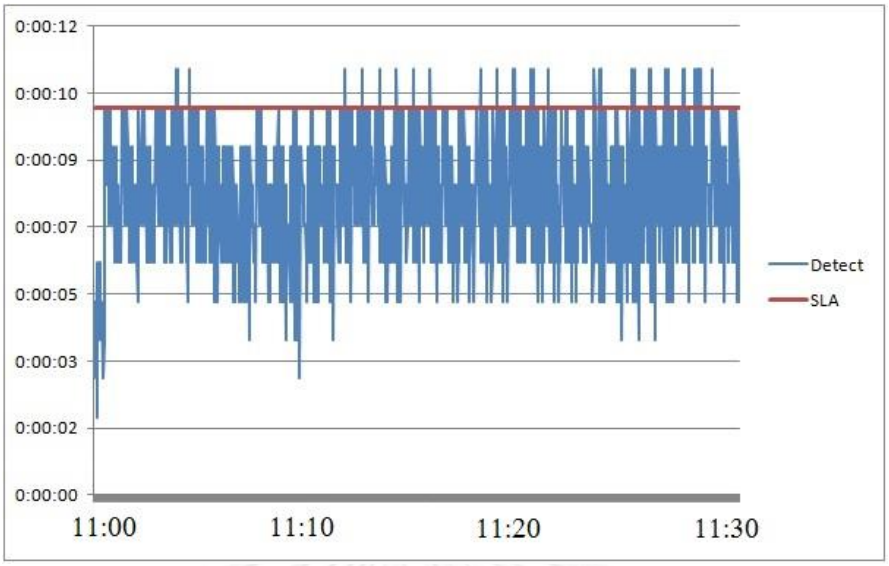
ในการทดลองนี้จะนำทฤษฎีทางด้านสถิติมาประยุกต์และคาดการณ์ถึงสิ่งที่อาจเกิดขึ้น โดยพิจารณาถึงขีดจำกัดของ Input rate ที่เหมาะสมสำหรับ Collector 1 ตัว บนสภาพแวดล้อมในการทดลองที่กล่าวไปแล้วในบทที่ 3 โดยทำการจำลองข้อมูล Log ในแต่ละช่วง Input rate จะได้ผลการทดลองการ Detect Error ดังภาพที่ 4.12



ภาพที่ 4.12 Histogram แสดงประสิทธิภาพการตรวจจับในแต่ละช่วง Input rate

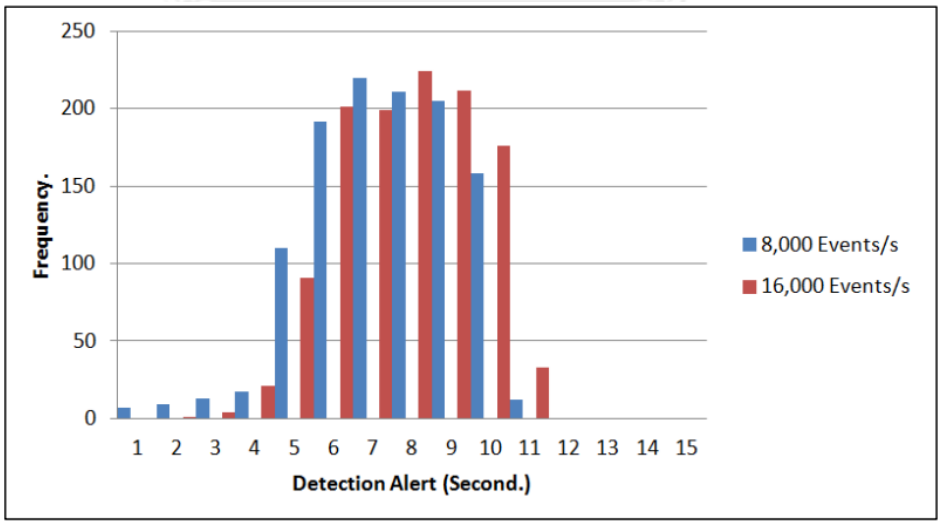
จากภาพที่ 4.12 แสดงถึงประสิทธิภาพการตรวจจับในแต่ละช่วง Input rate ซึ่งจะเห็นว่าในช่วงไม่เกิน 20,000 Events/s ระบบตรวจสอบจะยังมีประสิทธิภาพในการตรวจจับข้อมูลและมีลักษณะการตรวจจับเป็นการแจกแจงแบบโค้งปกติ แต่เมื่อ Input rate น า 20,000 Event/s จะเห็นว่ากราฟมีลักษณะเปลี่ยนไปเนื่องจากขีดจำกัดของระบบตรวจสอบดังที่กล่าวไปแล้วในบทที่ 4.2

ในการทดลองนี้ได้ทดลองการคำนวณหาความน่าจะเป็นของเปอร์เซนต์ที่จะเกินระยะเวลาที่กำหนด (Miss deadline ratio) ที่ Input rate รวมเท่ากับ 16,000 Event/s เป็นเวลา 30 นาที เฉลี่ย 10 ครั้ง จะได้ผลการทดลองคือ มีจำนวน Event ที่ผิดปกติเฉลี่ยทั้งสิ้น 500 Events โดยใช้โครงสร้างในการตรวจจับตั้งแต่เริ่มเกิด Log จนถึง SEC detect โดยผ่านการ Filter ได้ผลลัพธ์ดังกราฟ



ภาพที่ 4.13 กราฟแสดงประสิทธิภาพจากการตรวจจับความผิดปกติผ่านการ Filter

จากภาพ 4.13 แสดงประสิทธิภาพจากการตรวจจับความผิดปกติผ่านการ Filter (SEC-Filter) ซึ่งจะเห็นว่า หากระบบ Monitoring มีลักษณะการทำงานแบบ Hard Real-time โดยกำหนด SLA Deadline Constraint ไว้ที่ 10 วินาที จะทำให้เกิดการละเมิด SLA อย่างชัดเจน เนื่องจากระบบที่เป็น Hard Real-time จะไม่สามารถทนต่อการละเมิด SLA ได้เลย แต่เนื่องจากงานวิจัยนี้จะพิจารณาสิ่งที่เราสามารถตรวจจับได้ภายใต้การทำงานแบบ Soft real-time ดังนั้น จากกราฟที่ได้จึงขึ้นอยู่กับว่าระบบของเรานั้นสามารถยอมรับขอบเขตหรือเปอร์เซนต์ที่จะเกินระยะเวลาที่กำหนด (Miss deadline ratio) ได้มากเพียงใด



ภาพที่ 4.14 Histrogram แสดงประสิทธิภาพการตรวจจับ

จากผลการทดลอง เมื่อเราสร้างกราฟเป็นรูปแบบของ Histrogram จะได้ผลลัพธ์ดังภาพที่ 4.14 ซึ่งจะเห็นว่ามีลักษณะเป็น Normal curve และเราสามารถนำไปประยุกต์กับทฤษฎี Normal distribution ได้ โดยสามารถหาค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐานออกมาเป็นตารางได้ดังนี้

ตารางที่ 4.2 แสดงผลการทดลองแต่ละ Input rate

	Input rate	N	Mean	Std. Deviation	Std. Error Mean
Alert	8,000 Event/s	1154	7.47	1.830	.054
	16,000 Event/s	1154	7.82	1.668	.049

โดยที่;

- N คือ ค่าแสดงจำนวนครั้งของการ Detect แต่ละขนาด Input rate
- Mean คือ เวลาเฉลี่ยในการ Detect Alert (วินาที)
- Std. Deviation คือ ค่าส่วนเบี่ยงเบนมาตรฐานแสดงการกระจายของเวลาในการ Detect
- Std. Error Mean คือ ค่าความคลาดเคลื่อนมาตรฐาน

หากต้องการคำนวณหาโอกาสที่จะ Detect เกินเวลา Deadline ที่เรากำหนดไว้ สามารถคำนวณค่าความน่าจะเป็นได้จากพื้นที่ฝั่งขวาของกราฟ ตามทฤษฎีของ Normal distribution ดังนั้น จากสมการที่กล่าวในบทที่ 4 ซึ่งเมื่อเรากำหนด Deadline เราจะสามารถคำนวณโอกาสที่จะ Detect เกินเวลา Deadlineได้ดังสมการที่ 4.9

$$P\{T_{(i)} > D\} = 1 - P\{SLA(D)\}$$

<u>ตัวอย่างการคำนวณ</u> หากกำหนด Deadline (D) เป็น 10 วินาที ที่ Input rate เท่ากับประมาณ 16,000 Events/s เราสามารถคำนวณโอกาสที่จะ Miss Deadline ratio โดยใช้ Z-score จาก mean และ Standard deviation ตามตาราง ได้ผลลัพธ์ดังนี้

$$\begin{aligned} P\{T_{(i)} > 10\} &= 1 - P\{SLA(10)\} \\ &= 1 - P\{T_{(i)} \leq 10\} \\ &= 1 - P\{Z \leq \frac{10 - 7.82}{1.67}\} \\ &= 1 - P\{Z \leq 1.31\} \\ &= 1 - 0.9049 \\ &= 0.0951 \end{aligned}$$

จากผลการทดลอง จะเห็นว่าโอกาสของความน่าจะเป็นที่ Event log จะตรวจจับเกิน Deadline constraint ที่ 10 วินาทีเป็น 9.51% หรือ SLA จะการันตีถ้าหากระบบยอมรับ Miss Deadline Ratio ที่ไม่เกิน 9.51% ซึ่งจากตัวอย่างนี้ หากระบบกำหนดระดับข้อตกลงของ SLA ไว้ว่า “90% ของการตรวจสอบ จะต้องสามารถตรวจจับ Error จาก Event log ได้ภายใน 10 วินาที” จะ

ถือว่าการทดลองนี้ไม่ละเมิด SLA เนื่องจากระบบยอมรับได้ 90% นั้นหมายความว่า SLA ยอมรับขอบเขตหรือเปอร์เซนต์ที่จะเกินระยะเวลาที่กำหนด (Miss deadline ratio) ถึง 10% แต่ผลการทดลองอยู่ที่ 9.51% ซึ่งเมื่อกำหนด Deadline constraint เป็นค่าอื่นๆจะสามารถคำนวณความน่าจะเป็นของ Miss deadline ratio ได้ดังนี้

ตารางที่ 4.3 แสดง Miss deadline ratio ในแต่ละช่วง Deadline

Deadline(seconds) / Bandwidth(MB/s)	10	11	12	13	14	15
1	8.33%	2.68%	0.66%	0.13%	0.02%	0.002%
2	9.51%	2.87%	0.62%	0.09%	0.01%	0.001%

หาก SLA กำหนดว่ายอมรับ Miss deadline ratio ได้เพียง **5%** เราอาจต้องกำหนด Deadline เพิ่มเป็น **11** วินาที หรือมากกว่านั้น มิเช่นนั้นอาจทำให้เกิด SLA violation ได้

บทที่ 5
บทสรุปและแนวทางในการพัฒนาต่อ

5.1 บทสรุป

ในงานวิจัยนี้ทางผู้วิจัยได้ทำการพัฒนาและแก้ไขปัญหาที่เกี่ยวข้องกับระบบงาน Monitoring เพื่อรองรับการทำงานแบบเรียลไทม์โดยนำสถาปัตยกรรมเชิง Big data มาใช้ในการวิเคราะห์และหาความสัมพันธ์ของ Event log ที่เรียกว่า Log correlation เพื่อวิเคราะห์ต้นตอของสาเหตุและระบุปัญหาที่เกิดขึ้นจากข้อมูล log นอกจากนี้ยังนิยามความสัมพันธ์ของ Event model เพื่อหาความน่าจะเป็นในการกำหนดระยะเวลา (Deadline) สำหรับตรวจสอบระบบเรียลไทม์ที่มีความต้องการแบบ Soft real-time โดยประยุกต์หลักทางสถิติมาคำนวณความน่าจะเป็นที่จะละเมิดข้อตกลงที่นำมาเป็น Deadline Constraint ในระบบ ซึ่งในระบบที่มีลักษณะเป็น Soft real-time ในที่นี้ หมายถึงระบบจะสามารถทนต่อการละเมิด Deadline ได้บ้าง ตราบที่จำนวนของการละเมิด deadline ยังคงอยู่ในกรอบหรือขอบเขตที่ระบบสามารถรับได้

จากผลการทดลองแสดงให้เห็นว่าการเพิ่มโครงสร้างของระบบตรวจสอบเพื่อเพิ่มประสิทธิภาพในการตรวจจับสามารถออกแบบและกำหนดโครงสร้างเพื่อตรวจจับสิ่งผิดปกติในแต่ละส่วนตามระยะเวลาที่กำหนดได้หลากหลายรูปแบบ อีกทั้งยังสามารถใช้วิธีการวิเคราะห์ล็อกเชิงกฎเกณฑ์แบบกระจายเพื่อเพิ่มประสิทธิภาพให้ระบบสามารถตรวจจับได้เร็วยิ่งขึ้น และการนำทฤษฎีทางสถิติในส่วนของการแจกแจงปกติ (Normal distribution) สามารถนำมาประยุกต์ในการคำนวณหาความน่าจะเป็นที่จะตรวจจับได้ไม่เกิน Deadline constraint ได้

5.2 ข้อจำกัด

1. ในการทดลองนี้ได้ทำการ Monitor ข้อมูลล็อกที่เกิดขึ้นจริงบน Web Application เพียงแต่ข้อมูลที่ไหลเข้ามานั้นมีจำนวนไม่มาก จึงต้องทำการ Simulate ข้อมูลล็อกมาทดลองกับระบบ Monitoring ที่สร้างขึ้น
2. เนื่องจากเครื่อง Server ที่ใช้ในการทดลองนั้นสร้างขึ้นมาจากเครื่อง 7 เครื่อง ซึ่งถือว่าเป็นระบบขนาดเล็ก ดังนั้นเมื่อทำการทดลองกับปริมาณข้อมูลมากๆ บางครั้งเครื่องจะทำงานหนัก และส่งผลให้ประสิทธิภาพในการทดลองแย่ลงอย่างมาก

5.3 แนวทางในการพัฒนาต่อ

1. สร้างสภาพแวดล้อมในการทดลองให้มีขนาดใหญ่ขึ้น เพื่อรองรับปริมาณล็อกขนาดใหญ่ที่เกิดขึ้นจริง
2. ทดลองกับเครื่องมือ Monitoring และ Event Correlator ตัวอื่นๆเพื่อพิสูจน์ว่าการประยุกต์ทฤษฎี Normal distribution มาใช้กับการคำนวณ Deadline ที่เหมาะสมภายใต้ระบบแบบ Soft real-time นั้น ใช้ได้จริง

รายการอ้างอิง

1. Radar, O.R., Big Data Now: Current Perspectives from O'Reilly Radar. 2011: O'Reilly Media.
2. Siewiorek, D.P., R. Chillarege, and Z.T. Kalbarczyk, Reflections on industry trends and experimental research in dependability. Dependable and Secure Computing, IEEE Transactions on, 2004. 1(2): p. 109-127.
3. Simache, C. and M. Kaaniche. Availability assessment of SunOS/Solaris Unix systems based on syslogd and wtmpx log files: A case study. in Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on. 2005.
4. Avizienis, A., et al., Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, 2004. 1(1): p. 11-33.
5. Hanemann, A. and M. Sailer. A framework for service quality assurance using event correlation techniques. in Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings. 2005.
6. Hanemann, A.e.a., Towards a framework for IT service fault management. Proceedings of the European University Information Systems Conference, 2005.
7. Cristian, F., Understanding fault-tolerant distributed systems. Communications of the ACM, 1991. 34(2): p. 56-78.
8. John Bambenek, A.K., grep - Pocket Reference: the Basics for an Essential Unix Content-Location Utility. 2009: O'Reilly.
9. Oliner, A. and J. Stearley. What Supercomputers Say: A Study of Five System Logs. in Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on. 2007.
10. Cotroneo, D., S. Orlando, and S. Russo. Failure classification and analysis of the Java Virtual Machine. in Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on. 2006.
11. Hanemann, A. A hybrid rule-based/case-based reasoning approach for service fault diagnosis. in Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on. 2006.

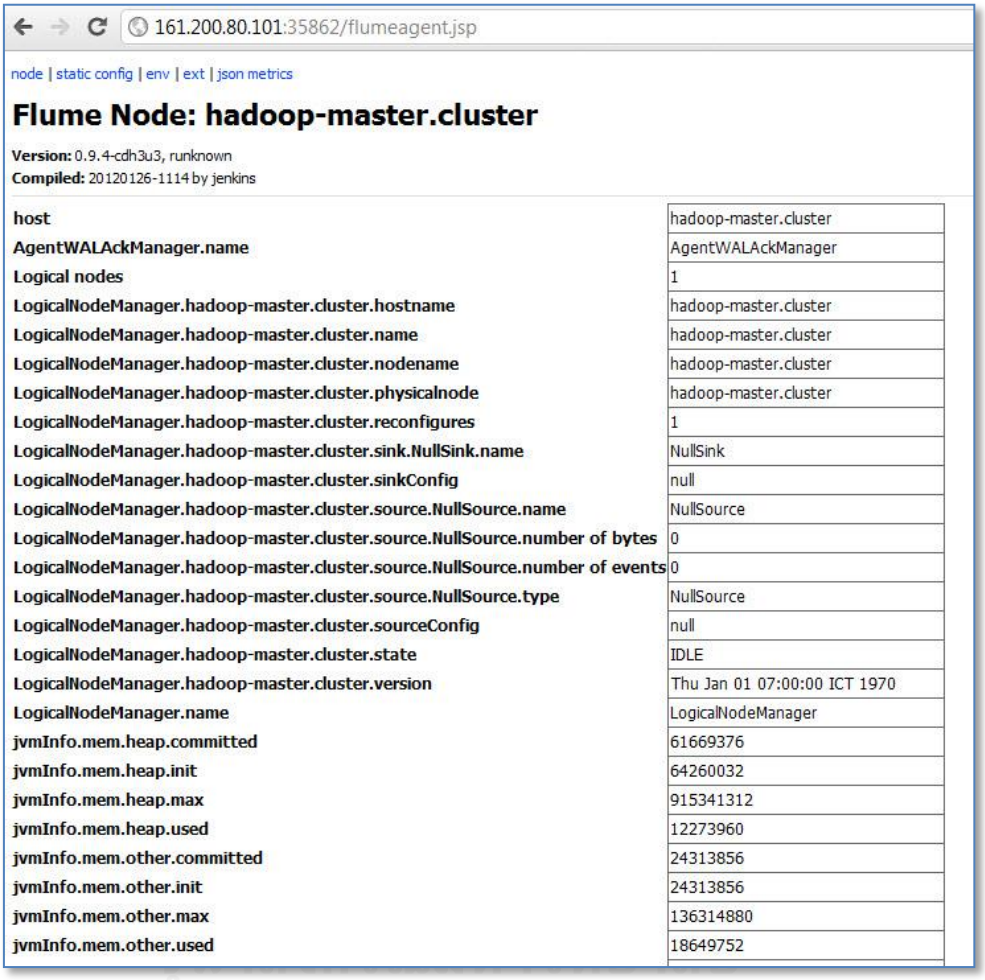
12. Cinque, M., D. Cotroneo, and A. Pecchia. A Logging Approach for Effective Dependability Evaluation of Complex Systems. in Dependability, 2009. DEPEND '09. Second International Conference on. 2009.
13. M. Cinque, R.N., A. Pecchia, S. Russo, Improving FFDA of Web Servers through a Rule-Based Logging Approach. Proceedings of the 1st International Workshop on Field Failure Data Analysis, Niagara Falls, NY, USA, 2008.
14. Hielscher, J., et al., A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing, in Towards a Service-Based Internet, P. Mähönen, K. Pohl, and T. Priol, Editors. 2008, Springer Berlin Heidelberg. p. 122-133.
15. Sammodi, O., et al. Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications. in Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual. 2011.
16. Hansen, J.P. and D.P. Siewiorek. Models for time coalescence in event logs. in Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on. 1992.
17. Gupta, M., et al., Discovering Dynamic Dependencies in Enterprise Environments for Problem Determination, in Self-Managing Distributed Systems, M. Brunner and A. Keller, Editors. 2003, Springer Berlin Heidelberg. p. 221-233.
18. http://en.wikipedia.org/wiki/Event_correlation.
19. http://en.wikipedia.org/wiki/Big_data.
20. Russom, P., Big Data Analytics. 2011.
21. http://en.wikipedia.org/wiki/Real-time_computing.
22. https://sites.google.com/site/mystatistics01/chapter1/probability-distribution.
23. Borthakur, D., The Hadoop Distributed File System: Architecture and Design. 2007, Hadoop Project Website.
24. http://archive.cloudera.com/cdh/3/flume-0.9.1+1/UserGuide.html.
25. Vaarandi, R. SEC - a lightweight event correlation tool. in IP Operations and Management, 2002 IEEE Workshop on. 2002.
26. Maren Salfner, M., A survey of online failure prediction methods. ACM Comput. Surv., 2010. 42: p. 1-42.
27. Michael R Grimaila, J.M., Robert F Mills, and Gilbert Peterson, Design and Analysis of a Dynamically Configured Log-based Distributed Security Event Detection Methodology. The Journal of Defense Modeling and Simulation, 2011(Applications, Methodology, Technology July 2012 9).

28. Wang, Y., et al., Monitoring and diagnosing software requirements. Automated Software Engineering, 2009. 16(1): p. 3-35.
29. Zawawy, H., K. Kontogiannis, and J. Mylopoulos. Log filtering and interpretation for root cause analysis. in Software Maintenance (ICSM), 2010 IEEE International Conference on. 2010.
30. Rabkin, A., and Randy Katz, Chukwa: A system for reliable large-scale log collection. Proceedings of the 24th international conference on Large installation system administration. USENIX Association, 2010.
31. Marz., N., Twitter's storm. https://github.com/nathanmarz/storm.
32. Neumeyer, L., et al. S4: Distributed Stream Computing Platform. in Data Mining Workshops (ICDMW), 2010 IEEE International Conference on. 2010.
33. Schmitz, A.H.a.D., Service–Oriented Event Correlation. In 11th International Workshop of the HP OpenView University Association (HPOVUA 2004), 2004(the MNM Service Model Applied to E–Mail Services).
34. Myers, J., M.R. Grimaila, and R.F. Mills. Log-Based Distributed Security Event Detection Using Simple Event Correlator. in System Sciences (HICSS), 2011 44th Hawaii International Conference on. 2011.
35. Mok, A.K. and L. Guangtian. Early detection of timing constraint violation at runtime. in Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE. 1997.
36. Abeni, L. and G. Buttazzo. QoS guarantee using probabilistic deadlines. in Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on. 1999.
37. Palopoli, L., et al. An Analytical Bound for Probabilistic Deadlines. in Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on. 2012.

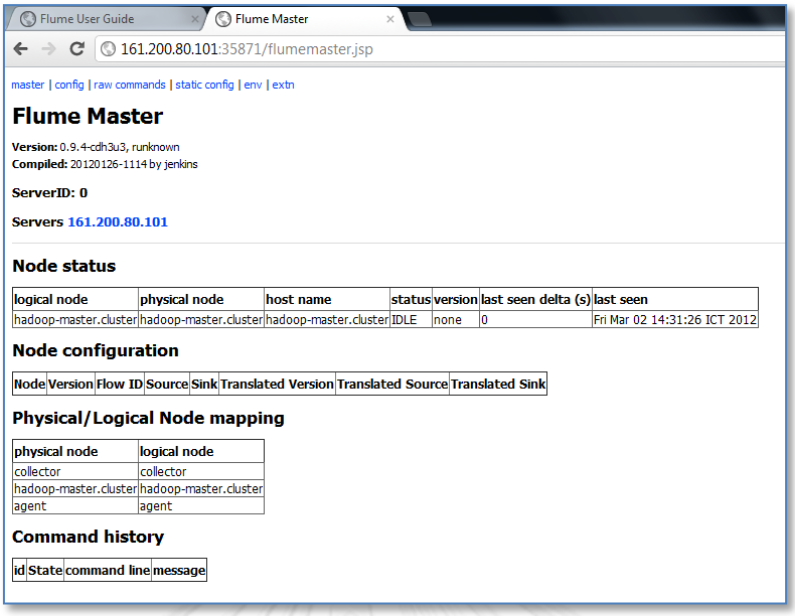
ภาคผนวก

ตัวอย่าง Interface Flume

วิธีการ Check status node อย่างหนึ่งก็คือเข้ามา web interface ซึ่งแต่ละ Node ก็จะแสดงข้อมูลของ Node บน machine ที่รันอยู่บนนั้น เมื่อ Node Agent มี Status เชื่อมต่อกับ Master ก็จะมีลักษณะดังภาพ

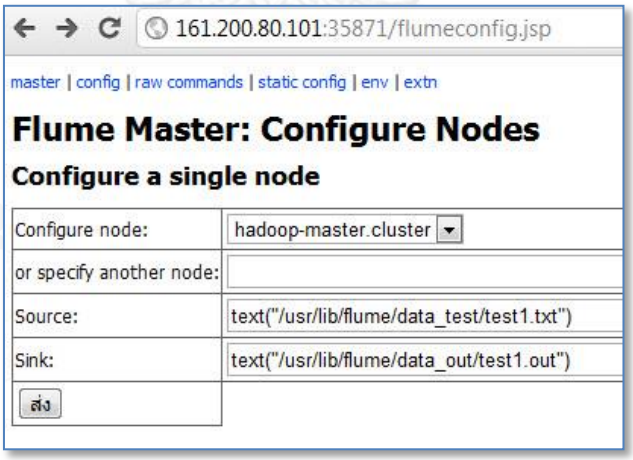


แสดง Status ของ Agent



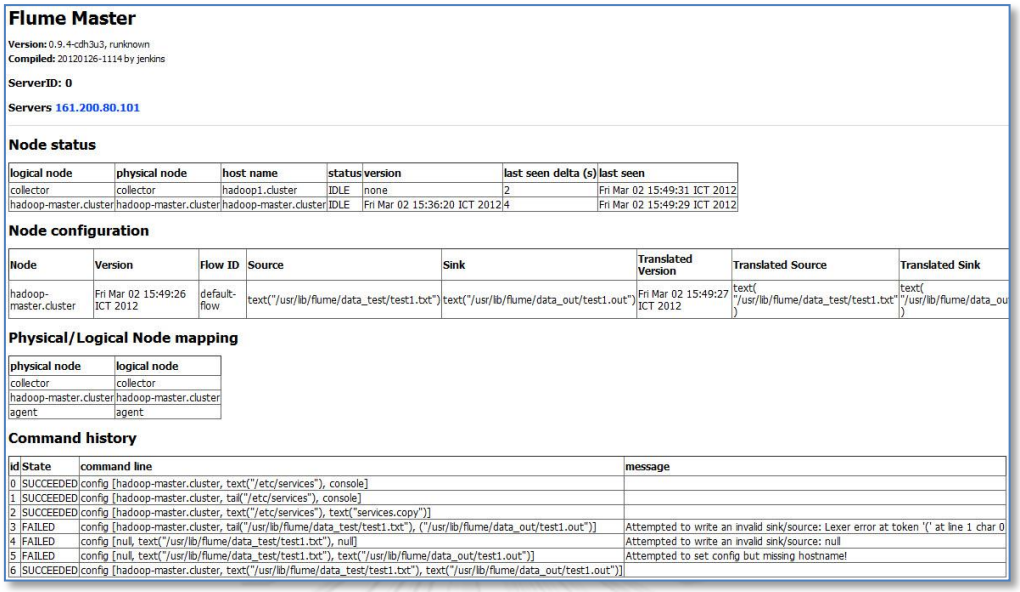
แสดง Status ของ Flume Master

เราสามารถ Configure ได้ตามต้องการแล้วก็สามารถเปลี่ยน Node data flow ได้ตามที่เราต้องการ โดยในหน้า Web page master ก็จะมี Link Configure ให้เราอยู่สอง Form เหมือนเราจะเรียกว่าเป็น web interface ในการ Setting Node's data flow ก็ได้ โดยที่เมื่อ Flume node มีการ contact กับ Master มันก็จะมีการแจ้งว่า data flow version ได้เปลี่ยนไปแล้ว ยกตัวอย่างดังภาพ



แสดงการ Configure ผ่าน Interface

จากภาพ เมื่อมีการสั่งคำสั่งผ่าน Source และ Sink เมื่อเรากด Summit หรือว่า "ส่ง" แล้วลองกลับมา Refresh ที่หน้า Master ก็จะเห็นว่าส่วน version มีการ Stamp เปลี่ยนเป็น Current time และ Source กับ Sink ก็จะถูก Update



Flume Master

Version: 0.9.4-cdh3u3, runknown
Compiled: 20120126-1114 by jenkins

ServerID: 0

Servers 161.200.80.101

Node status

logical node	physical node	host name	status	version	last seen delta (s)	last seen
collector	collector	hadoop1.cluster	IDLE	none	2	Fri Mar 02 15:49:31 ICT 2012
hadoop-master.cluster	hadoop-master.cluster	hadoop-master.cluster	IDLE	Fri Mar 02 15:36:20 ICT 2012	4	Fri Mar 02 15:49:29 ICT 2012

Node configuration

Node	Version	Flow ID	Source	Sink	Translated Version	Translated Source	Translated Sink
hadoop-master.cluster	Fri Mar 02 15:49:26 ICT 2012	default-flow	text("/usr/lib/flume/data_test/test1.txt")	text("/usr/lib/flume/data_out/test1.out")	Fri Mar 02 15:49:27 ICT 2012	text("/usr/lib/flume/data_test/test1.txt")	text("/usr/lib/flume/data_ou)

Physical/Logical Node mapping

physical node	logical node
collector	collector
hadoop-master.cluster	hadoop-master.cluster
agent	agent

Command history

id	State	command line	message
0	SUCCEEDED	config [hadoop-master.cluster, text("/etc/services"), console]	
1	SUCCEEDED	config [hadoop-master.cluster, tail("/etc/services"), console]	
2	SUCCEEDED	config [hadoop-master.cluster, text("/etc/services"), text("services.copy")]	
3	FAILED	config [hadoop-master.cluster, tail("/usr/lib/flume/data_test/test1.txt"), ("/usr/lib/flume/data_out/test1.out")]	Attempted to write an invalid sink/source: Lexer error at token '(' at line 1 char 0
4	FAILED	config [null, text("/usr/lib/flume/data_test/test1.txt"), null]	Attempted to write an invalid sink/source: null
5	FAILED	config [null, text("/usr/lib/flume/data_test/test1.txt"), text("/usr/lib/flume/data_out/test1.out")]	Attempted to set config but missing hostname!
6	SUCCEEDED	config [hadoop-master.cluster, text("/usr/lib/flume/data_test/test1.txt"), text("/usr/lib/flume/data_out/test1.out")]	

แสดงตัวอย่าง Flume Master หลัง Configure ผ่าน Interface

```
root@hadoop-master:/usr/lib/flume/data_out# cat /usr/lib/flume/data_test/test1.txt
1 one
2 two
3 three
4 four
5 five
6 six
7 seven
8 eight
9 nine
10 ten
root@hadoop-master:/usr/lib/flume/data_out# cat test1.out
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 1 one
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 2 two
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 3 three
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 4 four
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 5 five
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 6 six
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 7 seven
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 8 eight
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 9 nine
hadoop-master.cluster [INFO Fri Mar 02 15:49:29 ICT 2012] 10 ten
root@hadoop-master:/usr/lib/flume/data_out#
```

แสดงผลลัพธ์

Node status

logical node	physical node	host name	status	version	last seen delta (s)	last seen
161.200.80.101	161.200.80.101	hadoop-master.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	3	Wed May 09 00:34:09 ICT 2012
Andmebaas	Andmebaas	Andmebaas	ACTIVE	Thu May 03 00:40:12 ICT 2012	0	Wed May 09 00:34:12 ICT 2012
Kankrow	Kankrow	Kankrow	ACTIVE	Thu May 03 00:40:12 ICT 2012	0	Wed May 09 00:34:12 ICT 2012
hadoop-master.cluster	hadoop-master.cluster	hadoop-master.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	4	Wed May 09 00:34:08 ICT 2012
hadoop1.cluster	hadoop1.cluster	hadoop1.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	3	Wed May 09 00:34:09 ICT 2012
hadoop2.cluster	hadoop2.cluster	hadoop2.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	3	Wed May 09 00:34:09 ICT 2012
hadoop3.cluster	hadoop3.cluster	hadoop3.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	3	Wed May 09 00:34:10 ICT 2012
hadoop4.cluster	hadoop4.cluster	hadoop4.cluster	ACTIVE	Thu May 03 00:40:12 ICT 2012	4	Wed May 09 00:34:09 ICT 2012

Node configuration

Node	Version	Flow ID	Source	Sink	Translated Version	Translated Source	Translated Sink
161.200.80.101	Thu May 03 00:40:12 ICT 2012	default-flow	collectorSource(35853)	collectorSink("hdfs://hadoop-master.cluster/panya/syslogALL", "syslogALL")	Thu May 03 00:40:12 ICT 2012	collectorSource(35853)	collectorSink("hdfs://hadoop-master.cluster/panya/syslogALL", "syslogALL")
161.200.80.161	Thu May 03 00:40:12 ICT 2012	default-flow	null	null	Thu May 03 00:40:12 ICT 2012	null	null
Andmebaas	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
Kankrow	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
coll	Thu May 03 00:40:12 ICT 2012	default-flow	null	null	Thu May 03 00:40:12 ICT 2012	null	null
collector	Thu May 03 00:40:12 ICT 2012	default-flow	null	null	Thu May 03 00:40:12 ICT 2012	null	null
collector-master	Thu May 03 00:40:12 ICT 2012	default-flow	null	null	Thu May 03 00:40:12 ICT 2012	null	null
hadoop-master.cluster	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
hadoop1.cluster	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
hadoop2.cluster	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
hadoop3.cluster	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)
hadoop4.cluster	Thu May 03 00:40:12 ICT 2012	default-flow	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)	Thu May 03 00:40:12 ICT 2012	tail("/var/log/syslog")	agentSink("161.200.80.101", 35853)

แสดงตัวอย่าง Node status เมื่อทำงานร่วมกันหลาย Agent

ตัวอย่าง flume-site.xml

Master : /usr/lib/flume/conf/flume-site.xml

```
<configuration>
 <property>
<name>flume.master.servers</name>
<!--   <value>masterhost</value> -->
    <value>161.200.80.101</value>
    <description>This is the address for the config servers status
    server (http)
    </description>
  </property>

  <property>
    <name>flume.master.serverid</name>
    <value>0</value>
    <description> flume.master.serverid is the only Flume Master property
    that must be different on every machine in the ensemble.
```

```
    </description>


</property>
  <property>
    <name>flume.collector.event.host</name>
    <value>161.200.80.101</value>
    <description>This is the host name of the default "remote"    collector.
    </description>
  </property>


  <property>
    <name>flume.collector.port</name>
    <value>35853</value>
    <description>This default tcp port that the collector listens to
 in order to receive events it is collecting.
    </description>
  </property>
</configuration>
```

Hadoop1.cluster : /usr/lib/flume/conf/flume-site.xml

```
<configuration>

  <property>
    <name>flume.master.servers</name>
    <value>161.200.80.101</value>
    <description>This is the address for the config servers status
    server (http)
    </description>
  </property>

  <property>
    <name>flume.master.serverid</name>
    <value>0</value>
    <description> flume.master.serverid is the only Flume Master property
    that must be different on every machine in the ensemble.
    </description>
  </property>

  <property>
    <name>flume.config.master.addr</name>
    <value>161.200.80.101</value>
    <description>This is the address for the config servers status
server (http)  </description>
  </property>
</configuration>
```

ตัวอย่างการทดสอบ Collector

ทดสอบ tail file เพื่อให้แสดงบน console

```
Hadoop-master.cluster : tail(“/home/panya/testflume.txt”) | console;
```

ทดสอบ tail file เพื่อให้แสดงบน Console ของ Collector

```
Hadoop-master.cluster : tail(“/home/panya/testflume.txt”) |
agentSink(“161.200.80.101”,35853) ;161.200.80.101:collectorSource(35853) |
console ;
```

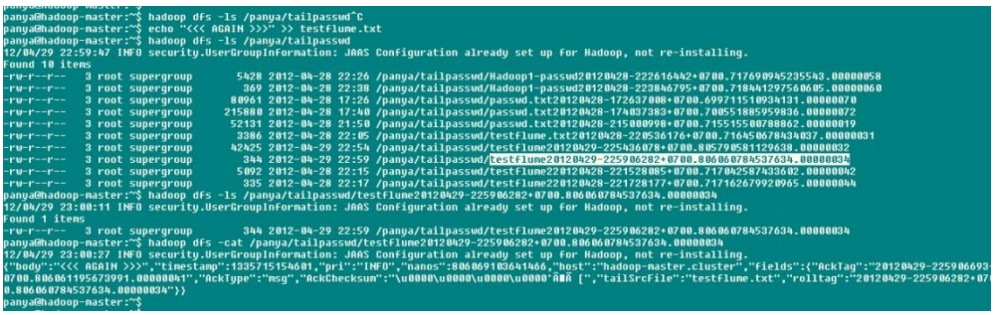
ทดสอบ tail file ลง HDFS

- 1 Agent

```
hadoop-master.cluster : tail("/home/panya/testflume.txt") |
agentSink("161.200.80.101",35853);
161.200.80.101 : collectorSource(35853) | collectorSink("hdfs://hadoop-
master.cluster/panya/tailpasswd","testflume");
```

- 4 Agents

```
hadoop-master.cluster: tail("/home/panya/testSEC/testSEC.log") |
agentE2EChain("161.200.80.101","161.200.80.103","161.200.80.104");
hadoop2.cluster: tail("/root/flumePerf/testSEC.log") |
agentE2EChain("161.200.80.103","161.200.80.104","161.200.80.101");
hadoop3.cluster: tail("/root/flumePerf/testSEC.log") |
agentE2EChain("161.200.80.104","161.200.80.101","161.200.80.103");
hadoop4.cluster: tail("/root/flumePerf/testSEC.log") |
agentE2EChain("161.200.80.101","161.200.80.103","161.200.80.104");
161.200.80.101: collectorSource | collectorSink("hdfs://hadoop-
master.cluster/panya/testSEC/","COLLECTOR-"); 161.200.80.103:
collectorSource| collectorSink("hdfs://hadoop-
master.cluster/panya/testSEC/","COLLECTOR2-"); 161.200.80.104:
collectorSource | collectorSink("hdfs://hadoop-
master.cluster/panya/testSEC/","COLLECTOR3-");
```



แสดงตัวอย่างไฟล์ที่เก็บบน HDFS

ตัวอย่างการเก็บ Syslog ผ่าน Database server มายัง HDFS

Andmebaas : tail("/var/log/syslog") | agentSink("161.200.80.101",35853) ;

Kankrow : tail("/var/log/syslog") | agentSink("161.200.80.101",35853) ;

161.200.80.101 : collectorSource(35853) | collectorSink("hdfs://hadoop-master.cluster/panya/syslog/%Y%m%d/","syslog-") ;

ตัวอย่างการกำหนด Output Format

syslog : a syslog like text output format

"May 20 18:17:02 hadoop3.cluster May 20 18:17:01 hadoop3 CRON[4403]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)"

log4j : a log4j pattern similar to that used by CDH output pattern.

"2012-05-20T19:17:02:+07:00 INFO log4j: May 20 19:17:01 hadoop3 CRON[4460]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)"

avrojson : JSON encoded date generated by avro

"{"body":"May 20 16:17:01 hadoop3 CRON[4319]: (root) CMD (cd / && run-parts --report/etc/cron.hourly)","timestamp":1337517343508,"pri": "INFO" ,"nanos":356167684305827,"host":"hadoop3.cluster","fields" :{"AckTag":"20120520-193542504+0700.356166680768408.00000064", "AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000\u0014m \u0018\u0005","tailSrcFile": "syslog","rolltag":"20120520-193735623 +0700.356278853637565.00000032"}}"

Example of a syslog error

{"body":"May 9 14:15:01 Kankrow postfix/sendmail[1904]: fatal: www-data(33): queue file write error","timestamp":1336606319202,"pri":"INFO","nanos":1440687297016862,"host":"Kankrow","fields":{"AckTag":"20120510063151981+0700.1440680076143562.00000032","AckType":"msg","AckChecksum":"\u0000\u0000\u0000\u0000Ã§sÂ‰Âˆ","tailSrcFile":"syslog","rolltag":"20120510063203688+0700.1697238190898202.00000020"}}

avrodata : Binary encoded data written in the avro binary format.

debug : a debugging format.

raw : output only the event body, no metadata

“May 20 07:17:01 hadoop3 CRON[4066]: (root) CMD (cd / && run-parts --report /etc/cron.hourly)”

```
      <property>
        <name>flume.collector.output.format</name>
        <value>avrojson</value>
        <description>This is the output format for the data written
to the
        collector.  There are several formats available:
          syslog - outputs events in a syslog-like format
          log4j  - outputs events in a pattern similar to Hadoop's
log4j pattern
          avrojson - this outputs data as json encoded by avro
          avrodata - this outputs data as a avro binary encoded data
          default - this is a format for debugging
        </description>
      </property>
```

แสดงตัวอย่างการกำหนด Format ที่ xml

ตัวอย่างการสร้าง Plugin เพื่อเพิ่มการทำงาน Filter

FilterSource.java

```java
/** Source **/
package filter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.cloudera.flume.conf.Context;
import com.cloudera.flume.conf.SourceFactory.SourceBuilder;
import com.cloudera.flume.core.Event;
import com.cloudera.flume.core.EventImpl;
import com.cloudera.flume.core.EventSource;
import com.cloudera.util.Pair;
import com.google.common.base.Preconditions;
/** Simple Source that generates a "Error" event every 3 seconds. */
public class FilterSource extends EventSource.Base {
  static final Logger LOG = LoggerFactory.getLogger(FilterSource.class);
  private String filTer;
  @Override
  public void open() throws IOException {
    filTer = "Error";
  }
  @Override
  public Event next() throws IOException {
    // Next returns the next event, blocking if none available.
    try {
      Thread.sleep(3000);
    } catch (InterruptedException e) {
      throw new IOException(e.getMessage(), e);
    }
    return new EventImpl(filTer.getBytes());
  }
  @Override
  public void close() throws IOException {
    // Cleanup
    filTer = null;
  }
  public static SourceBuilder builder() {
    // construct a new parameterized source
    return new SourceBuilder() {
      @Override
      public EventSource build(Context ctx,String... argv) {
        Preconditions.checkArgument(argv.length == 0,
            "usage: FilterSource");
        return new FilterSource();
      }
    };
  }
  public static List<Pair<String, SourceBuilder>> getSourceBuilders() {
    List<Pair<String, SourceBuilder>> builders =
      new ArrayList<Pair<String, SourceBuilder>>();
    builders.add(new Pair<String, SourceBuilder>("FilterSource", builder()));
    return builders;
  }
}
```

FilterSink.java

```java
package filter;  /** Filter ERROR from agents flow */
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.cloudera.flume.conf.Context;
import com.cloudera.flume.conf.SinkFactory.SinkBuilder;
import com.cloudera.flume.core.Event;
import com.cloudera.flume.core.EventSink;
import com.cloudera.util.Pair;
import com.google.common.base.Preconditions;
/** Write to a "filter.txt" file. */
public class FilterSink extends EventSink.Base {
  static final Logger LOG = LoggerFactory.getLogger(FilterSink.class);
  private PrintWriter pw;
  @Override
  public void open() throws IOException {    // Initialized the sink
    pw = new PrintWriter(new FileWriter("filter.txt"));
  }
  @Override
  public void append(Event e) throws IOException {
    // append the event to the output
    String str = new String(e.getBody());
    if (str != "" && str.contains("ERROR"))
    {    // here we are assuming the body is a string
    pw.println(new String(e.getBody()));
    pw.flush(); // so we can see it in the file right away
    }
  }
  @Override
  public void close() throws IOException {
    // Cleanup
    pw.flush();
    pw.close();
  }
  public static SinkBuilder builder() {
    return new SinkBuilder() {      // construct a new parameterized sink
      @Override
      public EventSink build(Context context, String... argv) {
        Preconditions.checkArgument(argv.length == 0,
            "usage: FilterSink");
        return new FilterSink();
      }
    };
  }
  public static List<Pair<String, SinkBuilder>> getSinkBuilders() {
    List<Pair<String, SinkBuilder>> builders =
      new ArrayList<Pair<String, SinkBuilder>>();
    builders.add(new Pair<String, SinkBuilder>("FilterSink", builder()));
    return builders;
  }
}
```

FilterDecorator.java

```java
/** Identify the source. */
package filter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.text.SimpleDateFormat;
import com.cloudera.flume.conf.Context;
import com.cloudera.flume.conf.SinkFactory.SinkDecoBuilder;
import com.cloudera.flume.core.Event;
import com.cloudera.flume.core.EventImpl;
import com.cloudera.flume.core.EventSink;
import com.cloudera.flume.core.EventSinkDecorator;
import com.cloudera.util.Pair;
import com.google.common.base.Preconditions;

/** Simple Decorator that prepends "HADOOP3" to every event body. */

public class FilterDecorator<S extends EventSink> extends EventSinkDecorator<S> {
  public FilterDecorator(S s) {
    super(s);
  }
  @Override
  public void append(Event e) throws IOException, InterruptedException {
    //String FilterBody = "HADOOP3 -- " + new String(e.getBody());
    //String FilterBody = new String(e.getHost()) + " -- " + Long.toString(e.getTimestamp())
    //+ " -- " + new String(e.getBody());
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    String FilterBody = new String(e.getHost()) + " -- " + sdf.format(e.getTimestamp()) +
                        " -- " + new String(e.getBody());
    EventImpl e2 = new EventImpl(FilterBody.getBytes(),

        e.getTimestamp(), e.getPriority(), e.getNanos(), e.getHost(),
        e.getAttrs());
    super.append(e2);
  }
  public static SinkDecoBuilder builder() {
    return new SinkDecoBuilder() {
      // construct a new parameterized decorator
      @Override
      public EventSinkDecorator<EventSink> build(Context context,
          String... argv) {
        Preconditions.checkArgument(argv.length == 0,
            "usage: FilterDecorator");
        return new FilterDecorator<EventSink>(null);
      }
    };
  }
  public static List<Pair<String, SinkDecoBuilder>> getDecoratorBuilders() {
    List<Pair<String, SinkDecoBuilder>> builders =
      new ArrayList<Pair<String, SinkDecoBuilder>>();
    builders.add(new Pair<String, SinkDecoBuilder>("FilterDecorator",
        builder()));
    return builders;
  }
}
```

การแก้ไข /usr/lib/flume/conf/flume-site.xml ในส่วน flume.plugin.classes

<value>filter.FilterSink,filter.FilterSource,filter.FilterDecorator</value>

```
<property>
  <name>flume.plugin.classes</name>
  <!-- <value>helloworld.HelloWorldSink,helloworld.HelloWorldSource,helloworld.HelloWorldDecorator</value> -->
  <value>filter.FilterSink,filter.FilterSource,filter.FilterDecorator</value>
  <description>Comma separated list of plugin classes</description>
</property>
```

flume-site.xml

การเพิ่ม /usr/lib/flume/bin/flume-env.sh

export FLUME_CLASSPATH=/usr/lib/flume/plugins/helloworld/filter_plugin.jar

ตัวอย่างคำสั่งการใช้งาน filter

hadoop4.cluster: tail("/root/flumePerf/testSEC.log") | agentSink("161.200.80.104");

hadoop2.cluster: tail("/root/flumePerf/testSEC.log") | agentSink("161.200.80.104");

hadoop3.cluster: tail("/root/flumePerf/testSEC.log") | agentSink("161.200.80.104");

161.200.80.104: collectorSource | [{ FilterDecorator() => FilterSink() },
collectorSink("hdfs://hadoop-master.cluster/panya/testSEC/","testSEC-")] ;

การตรวจจับ Pattern โดย SEC ผ่าน HDFS

- ใช้ FUSE เพื่อ Mount HDFS ภายใต้ path /tmp/hdfs ด้วยคำสั่ง

$sudo hadoop-fuse-dfs dfs://hadoop-master.cluster:8020 /tmp/hdfs

```
import java.io.*;
import java.util.*;

public class LogStreamOut
{
    private static final String READ_DIR = "/tmp/hdfs/panya/exp";
    private static final String ARCHIVE_DIR = "/tmp/hdfs/panya/exp/log_archive";
    private static final String FILENAME_PREFIX = "error";
    private static class LogFileNameFilter implements FilenameFilter
    {
        @Override
        public boolean accept(File dir, String name)
        {
            return name.startsWith(FILENAME_PREFIX);
        }
    }
    public static void main(String args[]) throws Exception
    {
        File archiveDir = new File(ARCHIVE_DIR);

        // Checkes whether archive directory is available to move log file to.
        while(!archiveDir.exists() || !archiveDir.isDirectory())
        {

            // Creates archive directory if it has not yet exist.
            boolean success = archiveDir.mkdir();
            if(success)
                System.out.println("INFO : Archive Directory is created.");
            else
            {
                System.out.println("ERROR : Fail to create Archive Directory.");
                System.exit(1);
            }
        }
        System.out.println("INFO : Archive Directory is now available.");

        while(true)
        {
            File folder = new File(READ_DIR);

```

```
40
41              // Searches for files that match the filename pattern.
42              File[] files = folder.listFiles(new LogFileNameFilter());
43              System.out.println("INFO : "+files.length+" file(s) found.");
44
45              // Overhead time which is "Read After Write delayed".
46              Thread.sleep(20000);
47
48              for(int i=0; i<files.length; i++)
49              {
50                  String fileName = files[i].getName();
51                  Scanner in = new Scanner(files[i]);
52
53                  boolean isReadyToRead = false;
54                  System.out.println("INFO : Reading "+fileName);
55
56                  // Reads line from log file and print to stdout
57                  // If no line was read from file, the file would be considered
58                  // as "Not Ready to Read".
59                  while(in.hasNext())
60                  {
61                      isReadyToRead = true;
62                      System.out.println(in.nextLine().trim());
63                  }
64                  in.close();
65
66                  // In case the file is "Ready to Read",
67                  // reads it and then moved to Archive Directory.
68                  // Otherwise, the file will reopen in next cycle.
69                  if(isReadyToRead)
70                  {
71
72                      boolean success = files[i].renameTo(new File(ARCHIVE_DIR+"/"+fileName));
73                      if(success)
74                          System.out.println("INFO : "+fileName+" is archived.");
75                      else
76                          System.out.println("WARNING : "+fileName+" cannot be archived.");
77                  }
78                  else
79                      System.out.println("WARNING : "+fileName+" is not ready to read.");
80              }
81          }
82      }
83
84  }
85
```

LogStreamOut.java

ประวัติผู้เขียนวิทยานิพนธ์

นายปัญญา กิตติพิพัฒนถาวร เกิดเมื่อวันที่ 14 พฤษภาคม พ.ศ. 2529 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาบัณฑิต หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ จากคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง เมื่อปี พ.ศ. 2552 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ณ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2554